THE BCS PROFESSIONAL EXAMINATION
Professional Graduate Diploma

April 2005

EXAMINERS' REPORT

Programming Paradigms

General
This module had a 90% pass rate. The highest mark was 75% and the mean was 53%. The general level of candidates' performance was therefore very pleasing, albeit not quite as good as last year. Questions 1 and 2 were by far the most popular and question 4 by far the least. There was no significant difference in the mean marks attained for the different questions.

Question 1 (Syllabus section 3)
1. Choose FIVE features of the object-oriented paradigm that you think are important. Describe what they are and discuss why you think they are important. Within your discussion, give examples of them, either with code or a diagram. **(5 x 5 marks)**

The answer should include a discussion of concepts such as inheritance, polymorphism, encapsulation, reuse, information hiding, etc. Candidates need to reflect on why they think they think the concepts are important. Examples from an OO programming language should be included to illustrate the points made, or a diagram if more appropriate (e.g., to show inheritance at the design phase).

Most candidates were adept at choosing five different object-oriented features and provided good discussions of concepts such as encapsulation and inheritance. Some candidates lost marks where they used different terms for similar concepts, e.g., encapsulation and information hiding, or polymorphism and overriding/overloading. Credit was obtained if different points were made for these similar concepts, but some candidates reiterated the same points, which did not achieve a high mark.

A higher mark was given to candidates who gave thought as to why the concept was important, whereas a lower mark was given if they provided just a description of the concept and did not provide an example of either code or a diagram.

Question 2 (Syllabus section 1)
2. A software house wishes to choose a single programming paradigm for all future developments. The company specialises in both database applications and artificial intelligence projects. It is faced with the variety of programming paradigms now available: logic programming, functional programming, imperative programming or object-oriented programming, any of which could be chosen as the single programming paradigm.

Choose TWO different programming paradigms and evaluate their strengths and weaknesses for such a company. Within your discussion explain why it might be difficult to choose a single paradigm for all future developments.

(25 marks)

Candidates were expected to give a description of the characteristics of the two chosen paradigms, with an evaluation of their strengths and weaknesses.

Within their discussion they needed to discuss how the paradigm would be appropriate for the software house. A good answer would have related the features of the paradigm to these particular types of applications. For example, imperative languages are more general-purpose and can be used for a variety of applications, whereas non-imperative languages, including functional and logic languages, are targeted at more specialised programming applications so would be suitable for AI but not for database applications. An O-O language might be appropriate for both.

A basic answer would have described two paradigms, with some evaluation. To get a good mark, some discussion of what type of system the paradigm is aimed at was required. A language good for database applications may not be appropriate for AI ones, which means that the search for the perfect paradigm is unlikely to be successful. One may either pick a language that is mediocre for the two different types of systems, or one that is good for one application but not the other.

Most candidates chose the object-oriented and logic programming paradigm as their two approaches. Quite a number of candidates reiterated the same points from question one again, which got little credit, unless they had tailored the answer to this question and discussed the advantages and disadvantages of each approach.
Quite a number of candidates provided good reasons for choosing the logic approach for AI applications, whilst some just described the features of logic programming (quite often lengthy descriptions of Horn clauses and Prolog examples), which did not gain much credit, unless it was to provide examples of why the approach would be good for such a company.

This question was not a "tell me all you know about object-orientation and logic programming" and candidates who did this gained low marks. Those who thought about the strengths and weaknesses of each approach got the higher marks.
Most candidates were able to provide convincing reasons why picking just one approach would be difficult, since the AI and database projects have such different requirements, neither an object-oriented language, such as Java nor a logic language, such as Prolog would be entirely suitable for both types of applications. The better candidates also provided arguments for which approach should be chosen if the choice was forced. Most favoured an object-oriented language in this situation, since it is more general purpose.

Question 3 (Syllabus section 4)

3. *a)* When using compound types within an applicative (functional) programming language, constructors, pattern matching and recursion all have a role. Using illustrative examples, discuss their roles within a functional programming language with which you are familiar. **(15 marks)**

b) Within the context of functional programming, lazy evaluation offers significant advantages over that of eager evaluation. Distinguish between these types of evaluation, and describe what these advantages are.

(10 marks)

This question addressed syllabus section 4. The selection of two different aspects of functional programming thus allowing candidates to demonstrate their breadth of understanding of the issues involved.

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

In general the answers to part (a) were very good and a few were excellent. There was a good use of examples to illustrate constructors, pattern matching and recursion, and these examples were based upon more than one functional programming language. In answer to part (b), candidates were expected to define the terms used, and to 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 may have related to storage, or time-based outcomes. Credit was given for any arguments correctly put.

There were many good answers to part (b), and some very good illustrative clarification examples. The majority of responses focused upon the time-based issues associated with lazy and eager evaluation, and only a few answers highlighted the evaluation storage implications.

Question 4 (Syllabus section 5)

4. *a)* What two major abstractions characterise logic programming, and how are these compromised in the implementation of practical logic programming languages? **(12 marks)**

b) Logic programming and other programming paradigms such as object-oriented programming and functional programming each have their own advantages. Many languages have attempted to capitalise upon the individual strengths of these programming paradigms by integrating them into one language. Present arguments for and against such integration. **(13 marks)**

This question addressed syllabus section 5. The selection of two different aspects of logic programming was intended to allow candidates to demonstrate their breadth of understanding of the issues involved. Credit was allocated roughly evenly; although a particularly good response to one part of the question obtained the corresponding weighting.

In part (a) of the question, candidates were expected to demonstrate their knowledge of the basic concepts of pure logic programming. The two major abstractions are: control statements are abstracted away, and assignment statements and explicit pointers are not used – unification is used instead. 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 have referred to logical or non-logical aspects, such as interface design or time dependency versus simplicity and consistency. No reasonable approach was rejected.

Whilst there were many reasonable responses to the initial question in part (a), and answers correctly explained the two major abstractions, in comparison the compromise and hence need for ‘extra-logical’ features element was poorly answered.
In answer to part (b), candidates were expected to present ‘for’ arguments that might have included 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.

Overall part (b) was not answered as well as part (a). A number of candidates did briefly discuss the very close relationship between the mathematics of functions (lambda calculus) and logic, and a few did mention in passing the differences in unification. Generally, candidates who got lower marks did so because they failed to provide sufficient detail.

Question 5 (Syllabus section 6)

5. *a)* “Synchronisation and communication are an important activity within programming languages that support concurrency.” Explain why this statement is true. **(10 marks)**

b) When a sequential program has executed its last statement, it terminates. What is the problem of program termination within the context of concurrent programming? Describe TWO solutions to this problem. **(15 marks)**

This question addressed syllabus section 6. The selection of two different aspects of concurrency was intended to allow candidates to demonstrate their breadth of understanding of the issues involved. The first part was more straightforward, and the second allowed more detailed discussion. More credit was therefore allocated to the second part.

In answer to part (a), candidates were expected to present their understanding of the need for synchronisation and communication within concurrent software. This need presents many problems, and candidates were expected to describe the importance of synchronisation and communication in the context of the models that have been developed to overcome these problems. Answers could have included a discussion of the shared memory model, the mutual exclusion problem, monitors, and synchronous message passing. Credit was given for any discussion correctly put.

There were a number of excellent answers to part (a). Many candidates included a discussion of the shared memory model, the mutual exclusion problem, monitors, and synchronous message passing. A number of candidates discussed Dekker’s algorithm. Some excellent examples were presented using Java concurrency constructs.

In answer to part (b), candidates were expected to introduce the subject of distributed termination and the associated problem of detecting termination or deadlock. Solutions to this problem could have included the use of signalling channels and process graphing, the Dijkstra-Scholten algorithm, the use of termination markers or distributed snapshots. Credit was given for each solution correctly put. Solutions using examples from real concurrency supporting programming languages attracted greater credit.
Part (b) attracted some very high marks with a number of candidates using the problem of the dining philosophers to illustrate the issues associated with program termination within the context of concurrent programming. Some candidates discussed the Dijkstra-Scholten algorithm and termination using markers.