THE BCS PROFESSIONAL EXAMINATION
The Professional Graduate Diploma

April 2000

EXAMINERS' REPORT

Programming paradigms

Question 1 – Answer Pointers

Covers syllabus section 1.

This question required the comparison of two programming types. 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 two chosen languages should have been given along with a discussion of what types of applications the languages were most suitable for. For example, imperative languages are more generally applicable and can be used for a variety of applications, whereas non-imperative languages, including functional and logic languages, 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.

The candidates were often good at explaining one type of language in detail, especially object-oriented languages such as C++, but were weak on comparing it to a second language. Some candidates mixed up concepts, even for OO, others gave weak examples of their chosen language type.

A lot of candidates lost marks by not giving examples of an appropriate application, or by gi ving superficial examples

Question 2 – Answer Pointers

Covers syllabus section 3.

The candidates were required to pick four different concepts from object-oriented programming and discuss why they are important. Concepts could include classes; encapsulation and data abstraction; methods and message passing; single and multiple inheritance; polymorphism, etc.

In their discussion the candidate were expected to say why they thought the concept was important to OO programming. Examples from an OO programming language such as C++, Smalltalk or Java should have been included to illustrate the concepts.

In this question the candidates often mixed up terms or misused them, in particular the terms abstraction, encapsulation and instantiation. Others gave no examples, or only for some of the concepts. Marks were lost if there was no discussion of why the concepts were important.

Question 3 – Answer Pointers

Both parts cover syllabus section 2.

In part (a) the candidate should have discussed the sort of tools available for developing a program, from writing the code to testing it. A discussion could include Interactive Development Environments (IDEs) or Interactive Development toolkits that provide a set of tools for writing the code, debugging, and testing it, plus configuration management utilities.

Part (b) required the candidate to reflect on the real usefulness of these tools. Within their discussion they should have indicated whether they thought the tools were important or not. For example, they could argue 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 the 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 say why they thought they were beneficial or not.

Most candidates answered part (a) successfully. A few lost marks by only giving a brief discussion of the appropriate tools. Some candidates showed signs of running out of time on part (b), or again only gave a brief discussion or did not reflect on how the tools improved programmer's productivity.

Question 4 – Answer Pointers

This question addressed syllabus section 4. The selection of two different aspects of functional programming was designed to allow candidates to demonstrate their breadth of understanding of the issues involved. Only one candidate attempted this question.

In part (a) of the question, the candidate was expected to define the nature of higher-order functions, and provide suitable illustrative examples of their role within a functional program. It was expected that the importance of such functions be discussed, in particular the map function, which should have highlighted a major difference between functional and imperative languages. Whilst the candidate introduced higher-order functions, mapping, a sorting algorithm illustration, and variable storage minimisation, the concept of functions having functions as arguments was not clearly presented. The map function was not used to illustrate the importance of higher-order functions. Nor did the candidate mention that such functions abstract away most of the control structures that are essential in imperative languages.

In answer to part (b), candidates were expected to demonstrate functional programming specific knowledge, and to understand how environments are relevant to imperative programming languages also. They were asked to comment upon similarities, or otherwise, and should have discussed issues such as bindings, structures, signatures and functors. The candidate's knowledge of an environment was illustrated solely with the aid of an imperative language example. Whilst credit was given for this, the discussion did not include any detailed reference to functional programming languages, and hence the issues such as bindings, structures, signatures and functors were not mentioned.

Question 5 – Answer Pointers

This question addressed syllabus section 5. The selection of two different aspects of logic programming should have allowed candidates to demonstrate their breadth of understanding of the issues involved. Only one candidate attempted this question.

In part (a) of the question, candidates were expected to demonstrate knowledge of the basic concepts of pure logic programming. Based upon familiarity with a real logic programming language, they were expected to understand that ‘extra-logical’ features are required, which are not in keeping with a pure logical approach. The reference to compromises was intended to stimulate a discussion of the need for such non-logical aspects as interface design or time dependency versus simplicity and consistency.

The candidate introduced the basic concepts of pure logic programming and the compromise with implementation issues such as specific search and order of computation rules. The candidate did not discuss other issues such as interface design, for example, output statements, the implementation of numerical computation, and the use of the cut to improve efficiency, etc.

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 could have included differences in unification, use of inference engine for logic programming, etc.

The candidate presented arguments for and against functional/logic programming integration, for which credit was given. However, these arguments were somewhat weak as the discussion tended to focus more upon a description of each programming paradigm per se, rather than upon the problems that would arise from their integration.