# THE BCS PROFESSIONAL EXAMINATION
**Professional Graduate Diploma**
**April 2003**

## EXAMINERS' REPORT

## Programming Paradigms

### General

There was a 100% pass mark for this module. The highest mark was 76% and the mean was 56%. The general level of candidates' performance was therefore satisfactory and the mean of 56% is a pleasing improvement on last year's mean of 51%.

It was notable that questions 4 and 5 proved much less popular than questions 1, 2 and 3. Specifically, questions 4 and 5 accounted for a total of 18 attempts, while questions 1, 2 and 3 accounted for 126 attempts.

### Question 1 (Syllabus section 2)

**1. a) Describe what tools are typically found in an Interactive Development Environment (IDE). (10 marks)**

**b) IDEs aim to improve the productivity of a programmer. Discuss how these tools can achieve this and also improve the quality of the code they produce. (15 marks)**

For the first part, candidates were expected to list the program development tools typically found in an IDE and describe what they do. Candidates might discuss debuggers, testing tools, linkers, configuration tools, loaders, screen painters, code generators, etc.

Part b required the candidate to reflect on the real usefulness of these tools. Within their discussion they should have indicated why they think the tools are important or not. For example, they might argue that they improve the speed and performance of the programmer, help with debugging, testing, etc. Disadvantages could include the encouragement of bad programming practices by relying on the tool to correct mistakes, rather than getting the programmers to check and test their code thoroughly themselves.

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

One the whole candidates made a good attempt at part a, often gaining high or full marks. In part b, some candidates were weak at discussing how the tools could improve productivity, or did not reflect on the fact that there could be bad points as well.

**Question 2 (Syllabus section 3)**

**2. Recently language design has focused on *"Programming in the Large"*. What concepts are found in object-oriented programming languages to support the development of large-scale applications? Include suitable examples.**

**(25 marks)**

Candidates were expected to consider the advantages and disadvantages of using an object-oriented (OO) language such as Smalltalk, C++ or Java. They should have should included a discussion of the concepts found in OO languages that aid the development of large systems, for example, inheritance, polymorphism, encapsulation, reuse, information hiding. They needed to reflect on what benefits or problems these concepts bring to program development and a good answer would have reflected on how it helps(or hinders) in a larger project. Examples from an OO programming language to illustrate the points made should have been included.

Candidates proved good at describing the concepts of object-orientation and gained an average mark for this if the descriptions were good. For a higher mark, they needed to reflect on which object-orientated features help with large-scale applications.

**Question 3 (Syllabus section 1)**

**3. "Every language is designed to solve a particular set of problems at a particular time according to the understanding of a particular group of people" (Bjarne Stroustrup, The Design of C++).**

**Choose two different programming paradigms and evaluate their strengths and weaknesses, illustrating your answer with examples from suitable programming languages. Within your discussion explain what particular set of problems they aim to address. (25 marks)**

A description of the characteristics of the chosen paradigms should have been given, with an evaluation of their strengths and weaknesses. Within their discussion candidates needed to discuss what types of applications the paradigm is most suitable for and a good answer would have related the characteristics of the paradigms to features of the particular types of applications.

For example, imperative languages are more general-purpose languages and could be used for a variety of applications, whereas non-imperative languages including functional and logical languages tend to be targeted at more specialised programming applications. OO languages may suit a company that uses OO methodologies for designing the system.

A basic answer would have covered the description of the two paradigms, with some evaluation. To get a good mark, some discussion of what type of system the paradigm is aimed at was required.

On the whole, candidates made a good attempt at this question, most being able to describe two different paradigms. Weaker candidates lost marks by only discussing one paradigm fully, or focusing on the strengths and ignoring the weaknesses. An average mark was given to candidates who discussed two paradigms but did not reflect on what sort of problems the particular paradigm was aimed at.

**Question 4 (Syllabus sections 4 and 5)**

**4. a) What are the two major abstractions that characterise logic programming? How are these compromised within the implementation of a practical logic programming language? (12 marks)**

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, they were expected to understand that 'extra-logical' features are required, which are not in keeping with a pure logical approach. The compromise discussion could refer to logical or non-logical aspects, such as interface design or time dependency versus simplicity and consistency. No reasonable approach was rejected.

**b) In a functional programming language, an expression is evaluated within the context of an environment. Discuss this statement commenting upon any similarities, or otherwise, that might exist between functional and imperative programming languages. (13 marks)**

Candidates were expected to demonstrate specific knowledge of functional programming, and to understand how environments are relevant to imperative programming languages also. Candidates are asked to comment upon similarities, or otherwise, and should have discussed issues such as bindings, structures, signatures and functors.

**Question 5 (Syllabus sections 6)**

**5. a) Why are process synchronisation and communication important activities within programming languages that support concurrency? (13 marks)**

Candidates were expected to demonstrate 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 might have included a discussion of the shared memory model, the mutual exclusion problem, monitors, and synchronous message passing. Credit was given for any relevant and correct discussion.

There were some good attempts at this section, and students were able to explain the importance of process synchronisation and communication with the context of concurrency supporting programming languages.

**b) A sequential program terminates when it has executed its last statement. What is the problem of program termination within the context of concurrent programming? Describe two solutions to this problem.**

**(12 marks)**

Candidates were expected to introduce the subject of distributed termination and the associated problem of detecting termination or deadlock. Solutions to this problem might 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 were awarded greater credit.

Generally the responses to this section were weaker than the previous section. Whilst some outline solutions were provided to the problem of program termination within the context of concurrent programming, the answers did not provide the level of required detail.  For example, no student mentioned the Dijkstra-Scholten algorithm.