BCS THE CHARTERED INSTITUTE FOR IT

BCS Higher Education Qualifications
BCS Level 6 Professional Graduate Diploma in IT

March 2018

EXAMINERS' REPORT

Programming Paradigms

General comments on candidates' performance

Future candidates are encouraged to expand the range of material studied in preparation for the course, looking to cover the full range of the syllabus.

There are some examples where candidates were not able to provide examples in certain languages. For the Programming Paradigms paper, it is desirable for candidates to be proficient in at least one language and have had the opportunity to learn and write some example programs in relevant languages for the other paradigms.

The following pages contain pointers about answers for the different questions, together with comments from the examiners.

Section A

A1

This question is about the **Nature of Programming Languages**.

a) Explain the essential characteristics and features for each of the following categories of programming language. Provide sample code to illustrate these features.

i. Imperative Languages. **(5 marks)**

ii. Declarative Languages. **(5 marks)**

iii. Scripting Languages. **(5 marks)**

b) Event-Driven Programming is often associated with programs for Graphical User Interfaces (GUIs) and Web Programming. Describe the meaning of the term Event-Driven Programming and provide code examples of how this style of programming works. Are all languages suitable for Event-Driven Programming?

(10 marks)

Answer Pointers

Part a)

For each of Imperative, Declarative and Scripting languages, there should be an appropriate explanation that includes some examples of the essential characteristics and

features. Stronger answers would distinguish between the different types and offer code extracts to illustrate the key ideas.

Imperative languages work by changing the values of variables. Imperative languages might be procedural in nature or be object oriented. The answer would expand on these ideas.

Declarative languages are used to express what computation should take place, without specifying the detailed steps required to complete the computation. Prolog and SQL are examples of languages where the problems are described, but the computation required is performed by the Prolog or SQL system.

A scripting language is one that is designed for specific tasks, e.g. command operations in a Unix or Windows shell, or can also be a more general-purpose language that is used to script specific actions, e.g. Python, Perl or PHP. Such languages are typically interpreted. Software applications may provide a scripting language to allow automation of a software package, e.g. VBA for automation of tasks in Microsoft Office on the Windows platform.

Syllabus Coverage: Nature of Programming Languages, 1.1 - 1.3

Part b)
The answer should provide detail about Event-Driven Programming, with at least one example to show how this code example works. This should cover the idea of an event and code that can respond to the event. Some notion of how the two are linked. For example, there might be a visual display of the GUI. A button might have an event property for when it is pressed. The property is linked to the code that will respond to the button press. Some assessment of whether all languages are suitable. The answer might comment that language styles such as imperative, scripting and functional are good candidates to support this technique, but declarative and logic languages are not such good candidates.

A strong answer may discuss that there might be arguments passed to event handlers. There could also be consideration of what makes a suitable language for event-driven programming.

Syllabus Coverage: Nature of Programming Languages, 1.5

Examiners' Comments

Part (a) covered imperative, declarative and scripting languages. This sub-question was almost universally done very well, and it was clear that candidates fully understood the key concepts and issues of all three. Many got full marks. Those that did not still achieved good marks per topic, and it was often just the lack of a small piece of code that stopped them getting full marks. Part (b) centred on event-driven programming and again, was well answered. Many candidates provided GUI diagrams to illustrate the user- interaction concept and most supplied a code snippet.

A2

This question is about **Programming Environments**.

a) What is an IDE and what are its key features and functions?

(5 marks)

b) Compare and contrast compilation and interpretation as software translation processes.

(10 marks)

c) ‘Testing is the same as debugging.’ Discuss this statement and explain whether or not you agree with it. In your answer, consider tools and techniques that might be used.

(10 marks)

Answer Pointers

Part a)
The term Integrated Development Environment (IDE) is defined along with a discussion of typical features – such as editing, translation, testing and debugging features. More advanced answers may go on to discuss features such as configuration and version control used within an IDE. The strong answers would typically add brief notes to explain the relevance of the features and functions within the IDE.

Syllabus Coverage: Programming Environments, 2.2.

Part b)
The answer would explain that compilation is a process that generates a totally separate (executable) output file that may be saved and run independently from the source file. Interpretation, however, is a process that does not generate such a logically and physically distinct output file, but rather translates the source to executable on the fly.

Answers should add further detail, including the concept of transforming source code written in a high-level programming language (the source language) into another computer language (the target language) usually having a binary form known as an executable program. Compilers versus Interpreters – where the term compiler means a program that produces a separate executable from the compiler (which may require a run time library). By contrast, an interpreter executes the program directly, translating each statement into a sequence of one or more subroutines already compiled into machine code – it does not generate a permanent separate executable file. It is the difference between translating a book in a library from one language to another (compiler) and translating a verbal exchange between two foreigners in the street (interpreter). The first produces a permanent (different) output object, the latter does not. Comments regarding the various stages of translation: lexical analysis, parsing, semantic analysis, code generation and code optimization would be credited.

Syllabus Coverage: Programming Environments, 2.1.

Part b)
The answer would discuss the two concepts of testing and debugging, highlighting the essential features of working to identify errors and investigating the causes of errors. There would be awareness of relevant tools and techniques to support these tasks and explanation about the need for these. For example: Types of testing – static (reviews, walkthroughs, or inspections) versus dynamic (executing programmed code with a given set of test cases), black box, grey box and white box testing, regression testing, user acceptance testing etc. Levels of testing – unit, integration, system. Coverage of debugging tools and the support that they can offer. Stronger answers could consider the role of automated testing and test harnesses and provide specific examples of the different types of tests, over and above describing them. The answer should offer a view of whether these are similar or different activities.

Syllabus Coverage: Programming Environments, 2.4, 2.5.

Examiners’ Comments
Part a) focused on the concept of an ‘IDE’ and its key features and functions. This was universally well done, and it was clear that the candidates were fully familiar with such an environment and all it had to offer. Part b) asked candidates to compare and contrast compilation with interpretation as software translation processes. This was generally well

answered but there was a much wider spectrum of the depth of detail provided. The evidence shows that most understood the essential differences but relatively few then went on to outline the detailed stages of each process. Finally, part c) asked candidates to address and compare the concepts of testing and debugging, including the tools and techniques that may be employed in both. Like part (b), there were a range of answers, although in general, most candidates did well and clearly differentiated testing, debugging and the various techniques and tools used in both.

A3

This question is about **Object Orientation**.

a) Within the context of the object-oriented (OO) paradigm, discuss the relationship between a class and an object, illustrating your discussion with appropriate examples. You should comment on the associated concepts of class/object attributes and behaviour and how they are represented. What specific OO concepts describe bringing together attributes and behaviour into a class and restricting (or removing) access to the implementation?

(10 marks)

b) A key element of the OO paradigm is the method. Begin by describing the purpose of the method concept before moving on to explain the different categories and types of method to be found in an OO programming language. Illustrate your discussion with simple examples.

(15 marks)

Answer Pointers

Part a)
The answer will address Classes as the abstract template or definition for some real-world concept or item, specifying data (attributes) and behaviour (methods) and objects as particular instantiations of a given class. The answer will build on this with further explanations of the encapsulation concept as linking data and operations on that data plus the information hiding concept - how it hides the internal implementation from the outside world and how the class is defined by its function, not implementation, all supported by at least one example.

Syllabus Coverage: Object Orientation, 3.1.

Part b)
The answer will identify the method as a block of logic that defines the behaviour of its host object to the outside world and how all activity on the object data (attributes) comes via methods. There will be detail about accessor methods (used to read data values of an object), mutator methods (used to modify the data of an object) and manager methods (used to initialize and destroy objects of a class - constructors and destructors). The stronger answers might discuss issues such as methods providing an abstraction layer that facilitates encapsulation and modularity – such as a bank-account class providing a getBalance() accessor method to retrieve the current balance, rather than directly

accessing the balance data fields - then later revisions of the same code can implement a more complex mechanism for balance retrieval, but this change in implementation, but not behaviour, can be hidden from the external world.

Syllabus Coverage: Object Orientation, 3.1.

Examiners' Comments
Part a) addressed the Object-Oriented (OO) paradigm, discussing the relationship between a class and an object. This was well done with most candidates providing a suitable diagram illustrating the class concept (attributes & methods) and many clearly-annotated code examples. Part (b) then asked candidates to describe the OO method concept before moving on to explain the different categories and types of method to be found in an OO programming language. This was less successful in that there was a lot of repetition from part (a), but many candidates did go into further depth on a range of relevant issues.

Section B

B4

This question is about **Logic Programming**.

a) A Prolog database about pets contains the following facts:

```
dog(rover).
dog(bert).
dog(snowy).
cat(fluffles).
cat(tom).
rabbit(thumper).

eats(tom, fish).
eats(bert, biscuits).
eats(fluffles, cream).
eats(fluffles, fish).
eats(rover, biscuits).
eats(rover, meat).
eats(snowy, meat).
eats(thumper, lettuce).
```

Provide an example of each of the following and describe how each would be used:

I. A ground query (no variables).

II. An existential/nonground query (using variables).

III. A conjunctive query (using a conjunction).

IV. A rule (with a head and a body).

(8 marks)

b) Describe the use of `findall/3` in Prolog and give an example of how it could be used with the example database in part a).

(7 marks)

c) Describe how negation-as-failure is used in Prolog. Give an example of a query using negation and explain how it is executed using the example database above.

(10 marks)

Answer Pointers

Part a)
The answer would attempt examples and descriptions for all of the parts of this question, for example:

i) `cat(tom).`

This asks whether the database can prove that `tom` is a `cat`. Prolog will respond with yes, because this fact exists directly in the database.

ii) `cat(X).`

This asks whether the database holds any atoms for which the cat predicate is true. Prolog will respond with the answer `fluffles`, and if prompted for more, then with `tom`, and then if prompted again, with `no`.

iii) `cat(X), eats(X, biscuits).`

This asks whether the database holds any `X` who is both a `cat` and eats `biscuits`. The query will only succeed if both parts succeed. The first part will instantiate `X` to the possible options in turn `(tom, fluffles)`, but the second part will fail in each case, and so the overall answer will be `no`.

iv) `carnivore(X) :- eats(X, meat).`

This is a rule with a head `(carnivore(X))` and a body `(eats(X,meat))`. We can read this as saying that if `X` eats meat then `X` is a carnivore (or `X` is a `carnivore` if `X` eats `meat`). We can add this rule to the database and then use carnivore as a predicate in further queries. Prolog would then respond to the query `carnivore(X)` with `rover` and then `snowy` and then `no`.

A strong answer would use appropriate technical terminology (such as 'atoms', 'instantiates', 'predicates'), and provide a description that indicates that the candidate understands how Prolog executes queries.

Syllabus Coverage: Logic Programming, 5.2

Part b)
`findall` allows us to collect a list of all objects that satisfy a given query/goal. It takes 3 arguments, the object to collect, the query to satisfy, and the list in which to collect the objects. Often the object is simply a variable.

We could use it as follows to query the given database.

```
findall(X, eats(X, meat), L).
```

`L` would be instantiated to the list `[rover, snowy]`

Syllabus Coverage: Logic Programming, 5.2

Part c)
Negation as failure is the use of the closed-world-assumption to determine negation. The closed world assumption assumes that all the facts that are true are either present in the database or derivable from the application of the given rules to the facts in the database. If we fail to determine that a fact is true, then we can derive that is must be false. A negation of a fact is true when we cannot determine that a fact is true. This is not the same as explicitly stating that a fact is false. The addition of new knowledge to a database may change the status of a fact or invalidate other previous reasoning. As an example, we could add the rule:

```
long_tail(X) :- \+ rabbit(X).
```

This would mean that `long_tail` would be satisfied whenever rabbit was not satisfied (so, examples for which `long_tail(X)` would be true are `rover, fluffles`, and all other dogs and cats. However, if we were later to add `guinea_pig(chomper)`, then `long_tail` would also be automatically satisfied by `chomper` (which does not have a `long tail`). Furthermore, any animals not present in the database at all would also satisfy this predicate, including for example an animal called `sid_snake`.

Any reasonable example would be accepted.

Examiners' Comments
Very few candidates attempted this question. The evidence shows that the answers provided did not demonstrate depth of knowledge of logic programming and were often unable to discuss these concepts in relation to the code example.

Logic programming is an area that candidates should be familiar with. Candidates are encouraged to look at this topic in more detail and where possible try out some of the basic concepts in Prolog as part of studying for this module.

B5

This question is about **Functional Programming**.

a) Some functions will have 'side effects'. In fact, some functions are written just for their side effects. Explain the term side effects and give an example of a function that has side effects as well as producing a value.

(8 marks)

b) Explain the terms domain and range, as they pertain to functions in functional programming. Choose a programming language that has a function for calculating the length of a string. What is the domain and range of this function? Is this a partial function or not? Explain your answer.

(7 marks)

c) Using a functional language of your choice, write a **recursive** function `naturalise` which should take a list of integers as its parameter and return a copy of the list of numbers except that negative numbers are replaced by zero.

For example, `naturalise [3,-5,6,-9,8]` should give the result `[3,0,6,0,8]`.

(10 marks)

Answer Pointers

Part a)
A pure mathematical function is a mapping from inputs to outputs. The output of the function is based only on the input, and no state is altered. On the other hand, a side-effecting function may or may not produce an output generated from calculations on the inputs, but will also cause side effects: interact with data or systems outside of the scope of the function. For example, it may update the value of a mutable variable, alter the display on a screen, input values from a keyboard or mouse, or write to a file.

An example of a side effecting function in Python:

```
def example(x, y):
    if x < y:
        print("x is the smallest")
        return x
    else:
        return y
```

The same example in Haskell:

```
example :: Int -> Int -> IO(Int)
example x y =
    if x < y
       then do
          putStrLn("x is the smallest")
          return x
       else
          return y
```

A stronger answer may indicate why this concept is important in programming, and the advantages of being able to program in a side-effect-free paradigm.

Syllabus Coverage: Functional Programming, 4.4

Part b)
Domain: the set of values that the function permits as its argument. Range: the set of values the function produces as a result. The domain for this function is any string input, including empty strings. The range will be integers greater or equal to 0. This is a total function as long as we are not restricted by memory: it will calculate an output for any possible input that is a string.

Syllabus Coverage: Functional Programming, 4.1

Part c)
An example solution in Haskell is show below. Other relevant solutions in a functional language would be accepted.

```
naturalise :: [a] -> [a]
naturalise []     = []
naturalise (x:xs) =
  if x < 0 then 0:naturalise xs else x:naturalise xs
```

Syllabus Coverage: Functional Programming, 4.2

Examiners' Comments

Of the candidates that attempted this question, there were some good answers, but there is evidence that many answers demonstrated limited understanding of functional programming. For Part b), some candidates started to define a function, but candidates were only expected to refer to a function that existed in a language the they were familiar with. For Part c), some candidates used a non-functional language, e.g. C or Java, however the question asked for an example using a functional language. Further, the examples in languages such as C or Java did not use recursion, so did not meet the other requirement in the question.

There are more functional languages available, e.g. Haskell and F#, and other programming languages are including aspects of functional programming. It is relevant to learn and understand this programming paradigm. Candidates are encouraged to look at this topic in more detail and where possible try out some of the basic concepts as part of studying for this module.

END