THE BCS PROFESSIONAL EXAMINATION
Certificate

April 2000

EXAMINERS' REPORT

Software Development

General Comment
A large number of candidates showed poor examination technique. Many candidates did not seem to take into account the weights given to the questions in the two sections. Many, in fact, spent more effort on questions in section B (12 marks) than they did on questions in section A (30 marks). This could indicate that while the candidates were conversant with the basic principles of the subject area, they lacked deeper comprehension of the material. There were a number of examples where candidates appeared to identify a certain phrase (for example, user interface) and then write everything they knew about the phrase rather than read the question carefully and concentrate on the what was actually requested (for example, differing requirements of user interfaces for different users).

Section A

Question 1.

Marking Scheme

a) A brief method of demonstrating the changes in the contents of the important memory locations is essential here. A possible way of expressing the dry run is :-

line no.	temp	decnum	rem	octnum	digit	instruction
1	0	?	?	?	?	
2	0	123	?	?	?	READ
3,4	0	123	3	?	?	
5	3	123	3	?	?	
6	3	15	3	?	?	
7	3	15	3	?	?	false->3
3,4	3	15	7	?	?	
5	37	15	7	?	?	
6	37	1	7	?	?	
7	37	1	7	?	?	false->3
3,4	37	1	1	?	?	
5	371	1	1	?	?	
6	371	0	1	?	?	
7	371	0	1	?	?	true->8
8	371	0	1	0	?	
9,10	371	0	1	0	1	
11	371	,,	,,	1	1	
12	37	,,	,,	1	1	
13	37	,,	,,	1	1	false->9

9,10	37	,,	,,	1	7	
11	37	,,	,,	17	7	
12	3	,,	,,	17	7	
13	3	,,	,,	17	7	false->9
9,10	3	,,	,,	17	3	
11	3	,,	,,	173	3	
12	0	,,	,,	173	3	
13	0	,,	,,	173	3	true->14
14	,,	,,	,,	,,	,,	output 173

As only 5 marks were allocated to the dry run, it was essential that candidates did not spend a disproportionate amount of time on it. The column method shown above is quicker than writing out in longhand what happens at every stage. It is recommended that only changes in contents of registers are written out; there is no need to repeat unchanging values, as is shown above for clarity. Dittos (") are acceptable provided their meaning is clear.

b) An alternative style of dry run is shown. This is more informative, but invariably takes longer to write out.

line	result
1	temp = 0
2	decnum = 16
3	REPEAT UNTIL decnum = 0 (false)
4	rem = decnum MOD 8 = 0
5	temp = 10 * 0 + 0 = 0
6	decnum = decnum DIV 8 = 2
7	UNTIL decnum = 0 (false)
3	REPEAT
4	rem = 2 MOD 8 = 2
5	temp = 0 + 2 = 2
6	decnum = 2 DIV 8 = 0
7	UNTIL decnum = 0 (true)
8	octnum = 0
9	REPEAT
10	digit = 2 MOD 10 = 2
11	octnum = 0 + digit = 2
12	temp = 2 DIV 10 = 0
13	UNTIL temp = 0 (true)
14	octnum = 2
15	

The choice of value for 'multiple of 8' is important. Values higher than 16 just add more cycles without revealing anything more informative about the result. (Those candidates who chose values such as 80 or 120 just made a lot of work for themselves.)

The output octal number should be 20, not 2 as produced. This happens because leading zeros are lost before the first reversed digit is stored. Very few realised this and knew how to correct the algorithm.

There are several amendments; the most plausible is to set 'temp' to 1 at the start, not 0, in which case the leading zeros are stored following the 1. An amendment is then required to line 13 : UNTIL temp = 1.

c) An acceptable PASCAL program derived from the corrected algorithm follows.

```
                                                        marks for line (or equivalent)
PROGRAM  decconv(INPUT,OUTPUT);
VAR  decnum, temp, rem, octnum : INTEGER;               3
BEGIN
  temp := 1;
  WRITELN('input decimal number');                      2 [require to have a prompt]
  READ(decnum);
 (* get reversed octal number *)                       2 [comment expected]
  REPEAT
        rem   := decnum MOD 8;
        temp := 10*temp + rem;                          3
        decnum := decnum DIV 8
  UNTIL  decnum = 0;
  octnum := 0;
(* reverse digits to get octal number *)               2 [comment expected]
  REPEAT
        digit := temp MOD 10;
        octnum := 10 * octnum + digit;                  3
        temp := temp DIV 10
  UNTIL temp = 1;
  WRITELN ('octal equivalent ='; octnum:3)              2
END.
```

Very few students included appropriate comments. It should be reinforced that the purpose of comments is to make clear the meanings of lines, or groups of lines, where this is not obvious from the code. Conversely, a few seriously over-commented their code. These extra comments are often superfluous as they add little to the meaning of the code.

Question 2

Marking Scheme

An acceptable data structure (for example, in Pascal) would be

```
type  element =  RECORD
                    name :   PACKED ARRAY [1..15] OF CHAR;
                    symbol : PACKED ARRAY [1..2] OF CHAR;
                                        {enumerated type also possible here}
                    relmass: REAL;
                    atnumber: 1..110;   {INTEGER also allowed, but subrange better as
                                         'atomicnumber' is never negative}
                    eltype:   BOOLEAN   {'true' for a metal, 'false' for a non-metal}
                 END;
```

5 marks were awarded for the data structure.

A possible algorithm would be

```
1       Declarations
2       OPEN FILE for reading
3       initialise variables
4       INPUT lower, upper for range of atomic numbers
5       VALIDATE lower, upper
6       LOOP WHILE NOT End-of-file (datafile) AND valid range DO
7               READ item
8               IF atnum >= lower AND atnum <= upper THEN
9               BEGIN
10                      IF metal THEN
11                              increment metal-count
12                      ELSE    increment non-metal-count
13              END
14      ENDLOOP
15      OUTPUT ('metals', metal-count, ' non-metals ' non-metal-count)
```

Development

2. This will be strongly language-dependent.
4.1 test if lower <= upper
4.2 test if lower >= 1 AND <= 110
4.3 test if upper >= 1 AND <= 110

Program code (17 marks) approx. mark allocation

```
PROGRAM ctelements (INPUT, OUTPUT, eldata);

CONST
        lower_bound   = 1,
        upper_bound   = 110;

TYPE element = RECORD  {as earlier – NO need to write it out again}

VAR     eldata : FILE OF element;
        item : element;                                                 2
        metalct, non-metalct, lower, upper : INTEGER:
        valid_range : BOOLEAN;

BEGIN
        WRITELN('count program for chemical element data');
        RESET(eldata);    (*appropriate code to open file *)            2
(* initialise variables *)
        metalct := 0;                                                   2
        non-metalct := 0;
        valid_range := true;
        WRITELN('input lower value of range');  READ(lower);            2
```

```
        WRITELN('input upper value of range');  READ(upper);
(* validate range values *)                                                    4
        IF NOT (lower <= upper) THEN
          BEGIN
                WRITELN('invalid range - lower value grater than upper value');
                valid_range := false;
          END
        ELSE
          BEGIN
                IF NOT ( (lower >= lower_bound) AND (lower <= upper_bound) ) THEN

                  BEGIN
                        WRITELN('lower range value outside set limits');
                        valid_range := false;
                  END;
                IF NOT ( (upper >= lower_bound) AND (upper <= upper_bound) ) THEN
                  BEGIN
                        WRITELN('upper range value outside set limits');
                        valid_range := false;
                  END;
          END;
(* process file of data *)                                                     4
WHILE NOT EOF(eldata) AND valid_range DO
BEGIN
        READ(eldata, item);   (* get item from file into single record *)
        IF (item.atnum >= lower)  AND  (item.atnum =< upper) THEN
          BEGIN
                IF item.eltype THEN (* corresponds to metal *)
                        metalct := metalct + 1
                ELSE  non-metalct := non-metalct + 1
          END;
END;
(* report results  *)                                                          1
WRITELN ('there were' metalct:2, ' metal and' non-metalct:2, 'non-metal elements present')
END.
```

Few students gave an initial algorithm and very poor development was typical. The question specifically asked for the development to be shown. Omission of the development was therefore penalised.

Small syntax errors in answers were not penalised. Marks were awarded primarily for the right type of instructions and comments in the correct place rather than absolute syntactic correctness.

Question 3

Marking Scheme

The first half of this question concerned the factors that should be taken into account when purchasing software and the relative importance of these factors. A good answer stated the relevant

factors that should have been taken into account when purchasing THIS software, coupled with a short discussion explaining each factor and discussing how important it was and why. Just providing a list of short bullet points (for example, 'cost') did not gain many marks. The question provided a comprehensive overview of what was being supplied and its application areas. Few candidates chose to tailor their answer to this and would have given exactly the same answer to a question along the lines of "What are the factors influencing your purchase of any software?". Some candidates even discussed points which were explicitly excluded; for example, they discussed the problem of not having source code listings. Again, when it came to the relative importance of the various factors, many candidates went through their list simply assigning low, medium or high importance to each of the points they had discussed, without explaining why they had assigned that particular priority.

The second half of the question was, in general, answered much worse than the first. The question asked for a test plan to be designed, yet many candidates did not include a test plan in their answer. Indeed, to many this seemed the perfect excuse to give a list of testing strategies with no discussion about the merits of each. The question specifically mentioned black and white box testing. Quite a few candidates chose to ignore that part of the question and some managed to get the definitions of the two the wrong way round. A full answer should have contained:

- A short discussion regarding when black box and white box testing is appropriate (possibly with a definition of each of these two types of testing strategies).
- A detailed test plan for the implementation of each task (again note, "for the implementation of each task") including whether they were black or white box tests.
- A brief dialogue regarding the reasons why the test plan was constructed in that way.
- The reasons why the tests were included.

Question 4

Marking Scheme

The first part of this question (15 marks) asked candidates to show their knowledge of current application software development tools in a PC environment. So, for example, a full (and accurate) description of a CASE or any IDE would have earned high marks. The second part of the question (15 marks) was looking for candidates' appreciation of how computing is developing; an awareness, for example, of the growing trend towards internet based solutions (and therefore the need to include support for this in development environments) or the need to include support for distributed teamwork in development tools.

In general this question was answered badly for several reasons. Many candidates, for example, appeared not to read the question properly. Either they provided a list of development tools but with no description (and the question specifically said describe the tools) or they described application software, not the tools required to develop it. Other candidates seem to have very little feel for the tools currently in use in a PC environment (again, this may point to a misreading of the question or perhaps indicates that candidates are selecting the wrong questions to attempt). Often only text editors, compilers and debuggers were discussed. Whilst these are all very necessary for software development, it does show a limited appreciation of tools available in a modern PC environment. Some described items which were not software development tools; for example prototyping or RAD - these are methods or techniques which make use of tools, but are not tools in their own right. Finally, when candidates looked to the future for these tools, they often just stated that they would cost less, be more powerful and more efficient. Answers like this are limited as they

do not display any thought or feel for the way in which applications (and therefore their development tools) are changing and will need to change over the next few years. A regular scan through current computing magazines would provide a lot of this information. One approach would have been to look at where current development tools are deficient and how current development requirements could be better addressed; for example, a move towards supporting cross platform projects and distributed solutions, or better integration of all aspects of the software development life cycle.

Section B

Question 5

Marking Scheme

A linked list can be used where a variable number of items is to be used in the data structure. If an array is used then nearly always a fixed block of memory must be assigned. If more items need to be put into an array than are foreseen then a 'boundary violation' will occur. If there are far fewer items, memory will be allocated but not used. With a linked list, only the memory actually needed for the items is used. With this data structure, it is easy to insert or delete single items from the list; no other items are moved. Insertion or deletion from an array requires all the others to be moved up or down. (6 marks)

An array could be used when the maximum number of elements is known and when random access to individual members is required, for example. This is easily gained by computing the subscript value for Member[subscript]. Thus sorting or searching operation are more efficient when an array is used. (6 marks)

Question 6

Marking Scheme

A compiled program is a block of executable code produced from source code by a compiler. It is not changed during a program run. Any syntactic source code errors are detected at the outset and reported on an error listing. If there are many compile time errors, propagation errors may be produced by early errors and likewise reported. This can make correction of errors later on in the source code difficult. This process is efficient if many runs of the program are required. (6 marks)

An interpreted program is run directly from the source code, each line (sometimes segment) of the code being directly translated as it is reached, before being executed immediately. Thus an error is reported individually, and execution stops where it occurred. Values of all the variables are directly available, unlike a compiled-program execution. Hence improved diagnostics are often available. This process generally results in repeated translation of the same line of code, which causes far slower execution speed than for compiled programs.

Often this process is used for small programs which can be developed interactively, hence it is often used for teaching. (6 marks)

Question 7

Marking Scheme

This was a very popular question. However, despite the question's popularity, many candidates scored very badly, showing little understanding of CASE tools and object-oriented programming (the two subjects of the question).

The first half of this question asked for brief notes on CASE tools. Specifically the question was looking for the candidates' understanding of what CASE tools are and how they might be used. As with the answers to some of the other questions, many candidates simply wrote down a list of features. Brief notes explaining what a CASE is, the purpose of CASE tools and how they are used would have added greatly to the simple list. (6 marks)

The second half of the question asked for brief notes on object-oriented programming. A large number of candidates insisted that this type of programming had to be visual, some went on to say that its only use, therefore, was for creating graphical user interfaces. A very high percentage of the candidates did not understand the concepts of objects and classes which is, of course, fundamental to this paradigm. Again, a list of unexplained (usually one word) features such as 'classes', 'polymorphism' and 'inheritance' was only part of what was required. A full answer might include a brief overview of the paradigm, definitions of a few key terms and an example. (6 marks)

Question 8

Marking Scheme

Simple diagrams with a sentence or two of narrative were sufficient for this question. In particular, candidates were expected to state the order in which the pointers were changed and show how many temporary pointers were required.

Eight marks were awarded for the general case of swapping the elements with two marks each for correctly commenting on the case when one of the elements was either the first or the last element.

Question 9

Marking Scheme

A popular question with the candidates but often poorly answered. The question asked candidates to consider two very different types of PC user and to discuss whether they need the same user interface or not. This was NOT an invitation to write down everything about interface design despite what some candidates seemed to think. The question required candidates to look at the requirements of the two types of user (inexperienced and professional) and show how user interface design principles apply to each. In order to draw a conclusion, they could then look at the amount of overlap between the two interfaces and in that way, determine whether or not two different interfaces were needed. Surprisingly, some candidates felt that for professional users, the interface was not important and so no interface design principles applied!

Question 10

Marking Scheme

Despite being very much a book work question, this question was very poorly answered. It was disappointing to see how little candidates knew. For example, when discussing data abstraction, many candidates confused it with information hiding and thought it was something specific to object-oriented techniques. Others described an example of an abstract data type (e.g. stack) but with no discussion on the process of data abstraction or why it was an abstract data type.

The candidates knew most about independent compilation, although a few did think it meant compiling programs without human intervention.

Very few candidates seemed to know much about software version control - only a handful of the several hundred answers mentioned change control or change management. Version control is not a method to avoid software piracy or a technique to prevent certain sectors of industry getting hold of particular software packages - definitions given on more than one answer. Marks were not given if the answer focussed only on the version numbers given to software (e.g. Visual Basic 6) unless reasons were given for why version numbers are needed and how they are used to ensure software integrity during and after maintenance/development.

Question 11

Marking Scheme

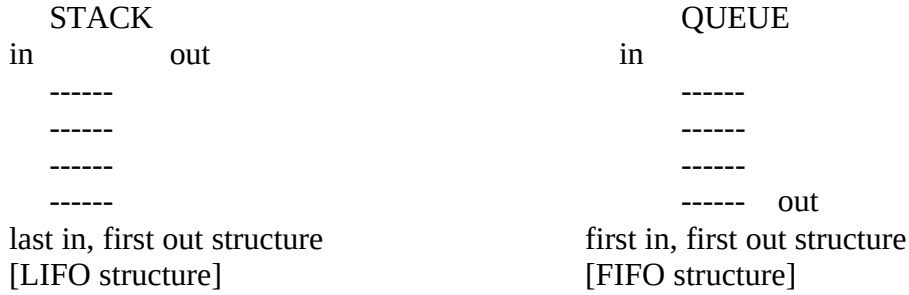
The first part of the question required little more than bookwork to describe a diagrammatic tool used to help **design** software and the second part required some reflection about the strengths and weaknesses of the tool. Some candidates answered this very well. However, a great many either had not read the question properly or did not understand what a diagrammatic software design tool is. For example, many candidates described the waterfall method (an approach to the software development life cycle), prototyping (a technique for developing software), project management tools, interactive debuggers and even programming languages. For those answers that described a diagrammatic design tool, many candidates did poorly on the discussion about its strengths weaknesses. Candidates need to be careful when commenting on the strengths and weaknesses of an approach or tool not to assign a weakness to that approach or tool when it was not designed to handle a particular aspect. For example, it is acceptable to comment that a Data Flow Diagram (DFD) does not show the relationships between the entities and that this aspect is handled by Entity Relationship Diagrams. This is not really a significant weakness of a DFD as they were not designed to handle this aspect of a system. It is not reasonable to criticise a tool for not doing something that it was never intended to do.

Question 12

Marking Scheme

a) stacks and queues (6 marks)

Both stacks and queues are dynamic data structures



This section was generally well answered. Some students did waste time writing implementations in code which was not required.

b) cohesion and coupling (6 marks)

cohesion concerns the relationship between data items within a particular module. It represents how tightly bound the internal elements of a module are to each other. It gives the designer a measure of whether elements belong in the same module. There are several levels of cohesion – thus coincidental, logical, procedural, sequential cohesion.

coupling measures the relationship of data between distinct modules. It thus attempts to capture how strongly two different modules are inter-connected. Generally coupling should be minimised.
For further details see Pankoj Jalote ‘An integrated approach to Software Engineering’.

Again this part was well answered which was pleasing to see as it is an important part of Software Engineering. Many students however spent far too long on this, writing detailed essays, as if it were a complete section A question, not half of a section B question, and worth only 6 marks. Candidates should realise that only 3 marks were available for cohesion and coupling individually, and judged the time accordingly.