THE BCS PROFESSIONAL EXAMIANTION
Certificate

Software Development

October 2001

EXAMINERS' REPOT

Software Development

General Comment

The answers to the program development type of questions would indicate that candidates have weak practical skills. An ability to design and develop a solution and create carefully thought out test data is just as important as knowing the syntax of a particular language.

QUESTION ONE

a) Develop pseudocode to process student examination marks from the file *'rawmarks' thus:*

Each student record contains a name (20 characters) and an identified (6 digits) followed by six integer marks expressed to the following criteria:
'PASS' : All six examination marks are passes
'RESIT' : Either 5 marks are passes and one mark is below %40% or 4 marks are passes with two marks lying between 35% and 40%.
'FAIL' : All other cases

An individual mark is deemed a 'pass' if it is 40 or more percent.

In addition to the processing outlined above, create two output files:
***'passlist'* containing the names and identifiers of only those candidates who have passed**
***'outfile'* a printer/character file containing the students' names, identifiers, exam marks, average mark and overall results.**

Separately count the number of pass, fail and resit candidates, and append this information to the end of the *'outfile'* with suitable captions.

You must show adequate development and give at least two stages of pseudocode development. Actual program code is not required. (24 marks)

b) Describe suitable data structures to use with the progam in a programming language of your choice. State which language you are using. (6 marks)

This question was not popular and in general it was poorly done.

Very few candidates knew how to develop a modest-sized problem from a specification to code via several stages of pseudo code. Many tried to write code straight from the specification, which is possible in this case, but is easier if developed systematically. The structure of nested 'IF' statements was invariably wrong.

Many candidates put down 'Pascal' as the target language but clearly have no experience of records or files. Very few attempted to make any distinction between the three files mentioned and used end-of-file instructions indiscriminately. Others showed an experience level appropriate to problems such as 'add a sequence of numbers and print the average' - quite inadequate for this problem. They ignored the instructions in the question and asked for the input items individually with prompts. A lot of writing for no marks! A few candidates did get the problem completely right.

b) Some candidates wrote about every data structure they could think of; e.g. trees, queues, stacks - appropriate or otherwise. Marks were awarded if the necessary data structures were given in a) as part of the program. If in doubt, candidates should not deliver a 'brain dump' of everything they know about data structures generally as it will not gain marks and just wastes their time.

Answer pointers:

A RECORD or STRUCTURE is needed to hold the details provided for each student such as:

```
        a_student  = RECORD
                        name : ARRAY [1..20] OF CHAR;
                        indent : INTEGER;
                        mark : ARRAY [1..6] OF INTEGER;
                     END
```

'rawmarks' will thus contain a sequence of these records thus
rawmarks : FILE OF a_student

A record for use with 'outfile' should also contain a character field to hold {pass/resit/fail} and a REAL field to hold the average mark.

As <u>three</u> files are mentioned in the question it is important to distinguish their use in the algorithm, e.g. END-OF-FILE(rawmarks), WRITE(outfile), WRITE(passlist). Vague use of READ, WRITE did not gain marks.

Indentation is helpful in showing the scope of BEGIN…END compound statements. It can be shown in several different ways, e.g. extended brackets, vertical lines.

It is easier to develop the algorithm in two or three stages. Stages left for later development can be marked with an * or similar. This avoids too much confusing detail at the first stage. Some methods suggest numbering these stages, useful as only those needing development are written out subsequently. The numbers need not be in strictly ascending order, and units of 10 allow for subsequent insertions.

This algorithm can be checked against the question to see if the required actions are correctly set down. In particular, that actions NOT asked for are not there. Details such as captions for tables, output of counters can be inserted at a later stage so that the initial algorithm is not cluttered with detail.

QUESTION TWO

The following program is intended to put the contents of three integer variables into descending orders:

```
Line No   code
  0       PROGAM order
  1       INTEGER VARIABLES num1, num2, num3
  2           PROCEDURE whatdo (INTEGER VARIABLES x, y)
  3           LOCAL INTEGER VARIABLES temp
  4           temp   ← x
  5           x      ← y
  6           y      ← temp
  7           END whatdo
```

```
 8          BEGIN {code for ‘order’ – top level}
 9          READ (num1, num2, num3)
10          IF num1 IS LESS THAN num2 THEN CALL whatdo (num1, num2)
11          IF num2 IS LESS THEN num3 THEN CALL whatdo (num2, num3)
12          WRITE (num1, num2, num3)
13          STOP
```

a) Dry run the above code with input values 1, 2 and 3. (12 marks)

b) What action is performed by the PROCEDURE whatdo? (2 marks)

c) Modify the given code so that the stated purpose of the program is achieved with <u>any</u> three input values. Clearly indicate where the modified code is placed in the supplied code, which need not be written out in full. (10 marks)

d) Specify a set of test data values sufficient to full test the code. (6 marks)

A popular question with the full range of marks awarded.

a) Candidates are familiar with this style of question and generally handled it well. Many did not realise that the parameters passed between procedure ‘whatdo’ and the top level MUST be of the Variable kind for the code to achieve its stated purpose. (Those who did not realise this and left ‘num1’, ‘num2’ and ‘num3’ unchanged were only penalised by 2 marks). It was particularly important to associate the given line numbers with the changes in variable/register contents.

b) The ‘whatdo’ code merely swaps the contents of the two variables ‘x’ and ‘y’. This was all that was needed to gain the 2 marks. One candidate wrote a page; nearly all supplied a paragraph. Those who wrote ‘swaps the contents’ gained the marks. A few thought it did a ‘sort’.

c) Those who got this far had the correct instructions, usually in the right place. An indication of why this was used was never given, but was desirable, particularly if the wrong variables were used by the candidate. The additional swap instruction necessary to place the variable contents in descending order can be derived from the result of the dry run and the stated purpose of the code. A few replaced the entire program with memorised code for a standard sorting method, not what was required. ‘Modify the given code’ meant ‘add or delete something’ not replace it all with something entirely different.

d) Guesswork was used here rather than discovering how to test each path through the code. Candidates need to learn how to provide systematic test data, not throw in as many numbers as come to mind to see if the code falls down. One mark was given for those who supplied real numbers, characters, enormous integers and the like. In these questions, should input error detection be required it will be clearly stated.

Answer pointers:

<u>Dry run of given program ‘order’</u>.

Important registers/variables have a vertical column. Line numbers are essential in showing the sequence of the run. Registers whose contents are as yet unset are best shown containing a ‘?’; not every language automatically sets them to zero. Registers unchanged on any particular line may contain “ (ditto) or left blank; it is not essential to repeat contents. The rightmost column should be used for clarification, registers not assigned a column or output.

Line number	num1	num2	num3	X	Y	temp	comments/output
1	?	?	?	?	?	?	Setup
8	“	“	“	“	“	“	Begin at top level
9	1	2	3	“	“	“	READ
10	“	“	“	“	“	“	IF{TRUE} call what
2	1	2	3	1	2	?	Parameters passed
3	“	“	“	“	“	“	Set up local variable
4	“	“	“	1	2	1	
5	“	“	“	2	2	1	
6	“	“	“	2	1	1	X ,Y now swapped
7	2	1	3	“	“	“	Values passed <u>back</u>
11	“	“	“	“	“	“	IF{TRUE} call whatdo
2	“	“	“	1	3	“	Parameters passed
4	“	“	“	1	3	1	
5	“	“	“	3	3	1	
6	“	“	“	3	1	1	X,Y now swapped
7	2	3	1	“	“	“	Values passed <u>back</u>
12							OUTPUT 2 3 1
13							STOP

1. The parameters ‘x,y’ <u>have to be of the variable type</u> which return values to the top level, otherwise the stated purpose of getting variable contents into descending order will never work.

2. The ‘comments’ column can be brief and need not contain the actual instructions - the line numbers suffice for that.

3. The stated purpose of getting the contents of num1, num2, num3 into descending order has not been achieved.

4. Procedure ‘whatdo’ SWAPS the contents of the variables x, y using ‘temp’ as an intermediate.

5. In order to complete the stated purpose, examination of the required dry run changed {1 2 3} to {2 3 1}. This suggests that a FURTHER swap of the contents of ‘num1’ and num2’ is needed which would place the values {1 2 3} in descending order {3 2 1}. Hence the instruction

```
IF num1 < num2 THEN CALL whatdo (num1, num2)
```

6. Is placed after line 11 and before line 12.

7. A full set of test data for the modified code will need SIX groups of 3 values, to test the pathways through the code. Thus appropriate values are :-

8.	1 2 3 (given)	2 3 1
9.	1 3 2	3 1 2
10.	2 1 3	3 2 1

There is NO advantage in using large or negative numbers here, or more than six groups of values.

QUESTION THREE

a) Specify and discuss the principles of multiple module program construction. (24 marks)

b) Describe the part that the client should play to carry out the task successfully. (6 marks)

Many candidates found sympathy with build, test and maintain. Many more only found sympathy with build alone. Answers were uniformly very short, typically one page, not at all reflecting any structured approach by candidates to earn 24 points by, say, identifying five or six separate things to say in order to address the breadth of the marks available.

Answer pointers:

Principles of modular construction

Design: high cohesion (purposefulness) with low coupling (low interconnectedness), black box function-focus ideas, layering, refinement

Implementation: separate compilation and programming languages that support this, name and reference to specific stores etc. obscured in OO ideas, dynamic space allocation to support efficient run-time operation in a multi-procedure program.

Integration: using teams, importance of integration testing and configuration management.

Maintenance: importance of regression testing, 'plug-replaceable' ability if no side effects, specific compilation sequence for rebuild and importance of configuration management. In part (b), candidates were asked to describe the role of the Client. Many gave no understanding of syllabus content, and said things like provide salary, provide office and infrastructure etc. Preferred answer indicated client input in TWO areas - requirements definition and validation testing. Any answer showing some awareness of these was generously rewarded.

Question 4

a) *Structures Programming* is a technique to reduce errors when programming. With respect to algorithm design, Structures Programming defines certain rules for the construction of algorithms. Describe at least THREE of these rules of Structured Programming, as they might be expressed for a modern programming language*1. (10 marks)

***b)* *Defensive programming* is a technique you can use to improve your program's ease of maintenance and robustness in use. Describe at least THREE techniques of Defensive Programming, as they might be expressed for a modern programming language. (10 marks)**

c) Compare and contrast the terms *Structured Programming* and *Defensive Programming* with reference to any modern programming language with which you are familiar. (10 marks)

*1 **a ‘modern’ language may be considered as any third-generation or visual programming language.**

Many candidates misunderstood structured programming as methodological and produced answers similar to question three on design issues.

Part b) was about Defensive Programming. The question took pains to describe what this meant and hinted very broadly at the expected answer. Candidates largely ignored this hint.

Answer pointers:

Structured programming has several features by which it can be recognised: single entry/single exit modules; controlled use of sequence, selection and iteration constructs; and very controlled use of jumps (e.g. pre-declared labels for compiler checking).

For robustness, Defensive Programming is a technique that simplifies file handling, uses validation checks on all inputs, and employs the ideas of no global variables to promote fewer failures at run time. For maintenance, the program text (names of variables, comments etc.) should clearly describe the algorithm and use of globals should be avoided. Any answers that evinced simple understanding were well rewarded.

Having answered the first two parts of the question, candidates only needed to demonstrate a thoughtful comparison of the two sections to gain high marks.

QUESTION FIVE

The number of combinations (C) from M items taken N at a time is given by

C= fac(M) / [fac(N) * fac (M – N)]

where fac(X) = X * (X – 1) * (X - 2) * … * 2 *1

Write a function to evaluate fac(X) and incorporate it in a progam to request input values M and N and calculate C. Provide appropriate captions and state which language you are using. (12 marks)

Nearly all candidates have seen the code for computing the factorial of an integer in programming books/classes and so this question was very popular.

Answer pointers:

Either the recursive method or the iterative method was acceptable. (Those who supplied both did NOT get extra marks!) In their zeal to put down something they knew some candidates forgot that it was associated with the calculation of the combinations (C), which needed an input of the number of items required (N) from the group (M) and an output of ‘C’ and so threw away marks.

A far worse error was to copy the definition of fac(X) directly into the function with the ellipsis (…) used in the code.

QUESTION SIX

An old computer program which processes car mileage and cost data accepts ONE pair of input data items per week. These are the amount of money spent (pounds sterling) and the price in pence per litre of the petrol bought. The program cannot accept more than one pair of data items for any one week and cannot be amended. However, in a week several petrol purchases may be made at different petrol prices per litre, not foreseen by the original programmers.

Develop the logic and write pseudocode which converts several such pairs of data items to an equivalent single transaction with an appropriate petrol price in pence per litre, so that the same volume of petrol would have been purchased.
(12 marks)

This short simple problem required a little thought. As it was clearly not bookwork it got very few answers. For those candidates with programming skills this question was a bonus as it could be solved and written down quite quickly

Answer pointers:

Few bothered to plan this answer and wrote code straight away and often made a fundamental error in calculating the volume of petrol. One candidate wrote that the original program was too old and <u>had</u> to be amended.

Although not suitable for anything other than very small problems, a significant number of candidates still used flowcharts. It would be of concern if candidates where being taught flowcharts as their only design method.

QUESTION SEVEN

Describe how to implement BOTH a stack AND a queue using

a) arrays (6 marks)

b) pointers (6 marks)

This was also popular, as is usual with bookwork questions. But so many thought all they had to do was state 'stack is a LIFO structure; queue is a FIFO structure' with nothing about implementation. Some candidates also confused the two. Very few had any ideas about implementing either structure using arrays.

Actual coding of the structures was not asked for, most candidates realised this. Just a few wrote very long answers with coding, and so spent a disproportionate amount of time on the question.

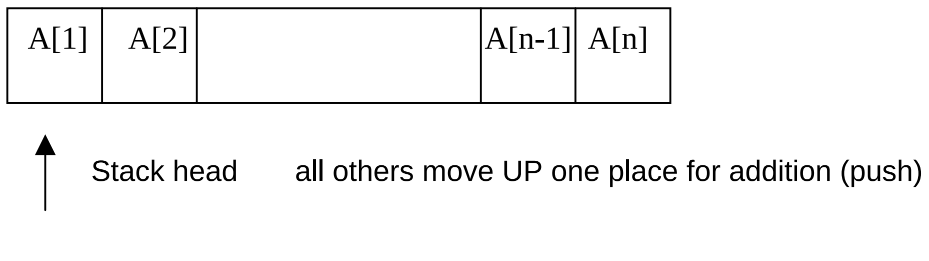
Candidates should not waste time on <u>repetition.</u> Something stated clearly by a diagram does not need a lengthy explanation in words as well. Repetition of an idea with different wording will not gain any more marks.

For only 12 marks candidates should have realised that the examiners were not looking pages of answers.

Answer pointers:

Successful candidates described a little of the techniques (arrays are contiguous, and static; dynamic space allocation is - well - dynamic but needs to be actively managed) and then identify the construction and use the features of the target data structures; i.e. building a stack or queue, accessing and inserting to the stack or queue, handling errors (overflow/underflow)

<u>Stack using arrays</u>

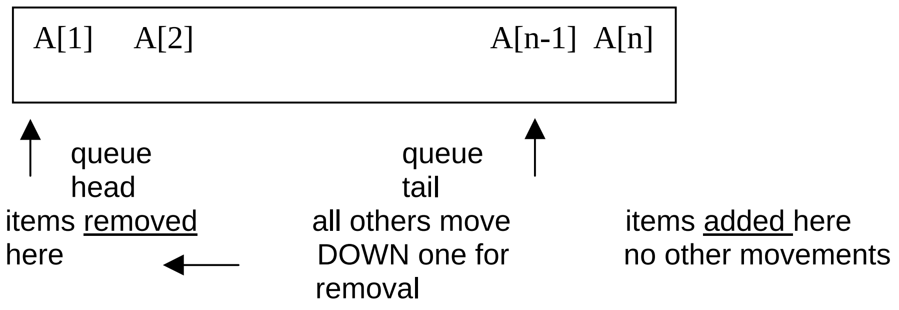


AND DOWN one place for removal (pop)

Stack tail

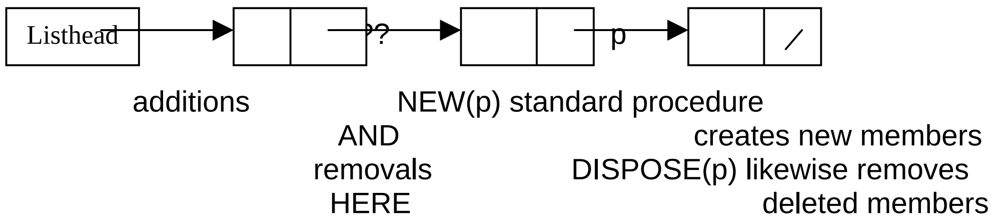
If the array is large a lot of processing is needed for each addition/removal

Queue using arrays

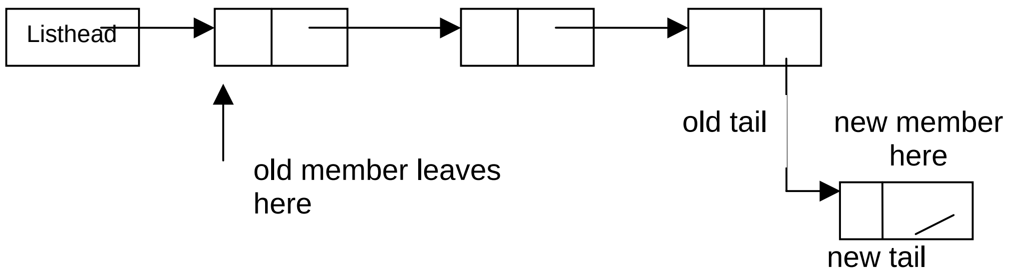


The array must be declared big enough for the maximum number of items ever likely to be needed.
This can be very wasteful of space. Some systems allow dynamic array declaration but the overhead is a lot of processing.

Stack using pointers.



Queue using pointers



QUESTION EIGHT

Describe the operation of a modern file/database access mechanism such as VSAM or ISAM. Include a description of deletion, updating and insertion of data, as well as a commentary on its strengths and weaknesses. (12 marks)

This was about the data structures for the indexes that support access to large databases. Index Sequential Access Method (ISAM) is a hybrid index that tries to deliver fast sequential processing such as printing of invoices as well as fast direct access to support, say, telephone or on-line queries.

VSAM is a more modern version of the same thing - Virtual Sequential Access Method.

Answer pointers:

ISAM and VSAM are not, in structure, like arrays but like trees. At each leaf there are several nodes describing the access into the database*.*

The nodes are sorted into key order, and distributed across the tree in a way to minimise the searching time. First fast tree **search** 'knows' the approximate location because **the keys are pre-sorted**. Within a leaf, a scan search is made to locate a particular key.

Insertion can be tricky if a node is already full. Then a new node is created and some keys unloaded into the new node in order to maintain the sorted-ness of the keys.

Deletion is less tricky unless it is deletion of the last key at a node, when the node can be removed completely as this will speed up the tree search.

Frequent **maintenance** is needed, to re-balance the tree and ensure it has equal 'arms' to avoid excessive tree searching down one or other 'long' arm. This process also involves redistribution of the keys to maintain capacity for growth without adding of further nodes, as much as is possible.

QUESTION NINE

You have been asked to plan the development of a website for a local small business. Describe the documentation you would specify as part of the project's deliverables. Give your reasons for the specified documentation and include any assumptions you make about the needs of the small business. (12 marks)

This was about documentation, but put in the context of a website development for a small business. Many candidates did not read the question carefully and thought it was about planning a website development.

Answer pointers:

The answer required the candidates to select the essential material that would be needed to keep such a simple website operational. Many candidates gave answers that would keep a mission critical strategic defence network operational.

Update and change - what **tools** were used and where are the **files** that constitute the data for the website.

Maintain and publish - upload procedure, **passwords**, **ISP** details, etc.

Any coherent awareness of 'horses for courses' to maintain a website was rewarded. Answers that needed all the data flow diagrams, and a manual to tell the users how to use the website, were clearly too detailed.

QUESTION TEN

Discuss the impact of software tools (such as web page generators, and application generators) on the testing plan of a small to medium sized commercial software project. Comment especially on the relative merits of black box and white box testing, the testing plan (whether top-down or bottom-up) and any other verification and validation activities that you consider important. Give your reasons. (12 marks)

This question was about testing in the context of tools. Many candidates ignored the context and gave the book about testing. Other candidates misinterpreted the context of development tools and discussed the use of tools in general to support the activity of testing.

The question gave candidates clear hints about the answer expected; the answer was to discuss three distinct areas; black box versus white box, bottom-up plans versus top-down plans, and validation and verification.

The question expected candidates to realise that tools were automatic generators, and the utility of white box testing would be much reduced. Likewise, the functional side of black box, integration testing and validation testing would be much enhanced leading to preference for top-down plans and validation events.

Many candidates realised the tools produced executable code automatically, and then brazenly stated that the programmer needed to white box check this code to ensure its correctness. Many candidates were even-handed in describing all the issues asked for and refused to come to any conclusion about what to do where development tools were being used which was acceptable.

Candidates who discussed tool support for the process of testing were rewarded according to the scope and coherence of their discussion. Often, the discussion was 'in parts' where tools were discussed, then, and quite separately, testing was discussed, and no integration was made.

Answer pointers:

White box versus black box: functionality increases in importance so black box is the main testing scheme. White box delivers very little added value because of the *automatic generation* nature of the tool.

Test planning bottom-up is not so useful for the same reasons. Top-down is more productive and useful.

Validation and verification highlights the need for user involvement.

QUESTION ELEVEN

Describe with suitable examples, what the term *abstract data type* means. Your answer should consider at least THREE different examples with illustrations of their use. (12 marks)

This question sought to find out if the candidate knew about the concept of abstract data types and its application.

Many candidates did know, and got good grades. Other candidates had no knowledge of this.

Answer pointers:

Essentially, an abstract data type collects together the code and data to implement a piece of functionality, concealing the implementation details and offering externally only the 'levers' to manipulate the structure - construction, manipulation, error querying. Three suitable examples were sought, and the answers were tested for their recognition that implementation details were not visible whereas construction, manipulation, error querying had to be available.

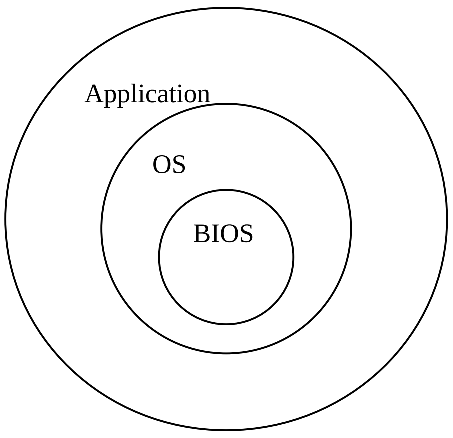
QUESTION TWELVE

Computer systems are often described using the metaphor of a set of concentric (onion-like) rings. State whether this is a useful idea and describe one computer system in this way to justify your answer. (12 marks)

Candidates were expected to draw on their own experience and to be able argue a simple case.

Answer pointers:

Some sort of picture model was expected to evince an understanding.



Following this, a description of some detailed knowledge that fitted this model was wanted. For example, BIOS, WINDOWS, ACCESS or UNIX, SHELL, APPLICATION.