THE BCS PROFESSIONAL EXAMINATION
Certificate

April 2004

EXAMINERS' REPORT

Software Development

General Comments

At the Certificate level, the examination covers the syllabus as a whole and asks candidates in part to describe what they know, and in part to apply what they know to specific situations. Overall, descriptions were well answered. Applications were not.

As has been written in previous reports, practice by marked homework is a better preparation for this paper than practical sessions at a PC/terminal. This paper is independent of computer language, because of the numerous languages taught world-wide but algorithm development is needed for any problem that is not trivial, and is moderately independent of language. Creating simple programmes on a PC to solve simple problems which need no algorithmic development suggests wrongly to students that getting the syntax right is the main aim (for only then will the program run).

This paper examines knowledge of programming by simple algorithms, not programming syntax. A database language should not be used for candidates' first experience of programming. Often the code is hidden from the user, or implied; even if visible, the translated complexities of database language syntax are too great for understanding how to develop algorithms in it.

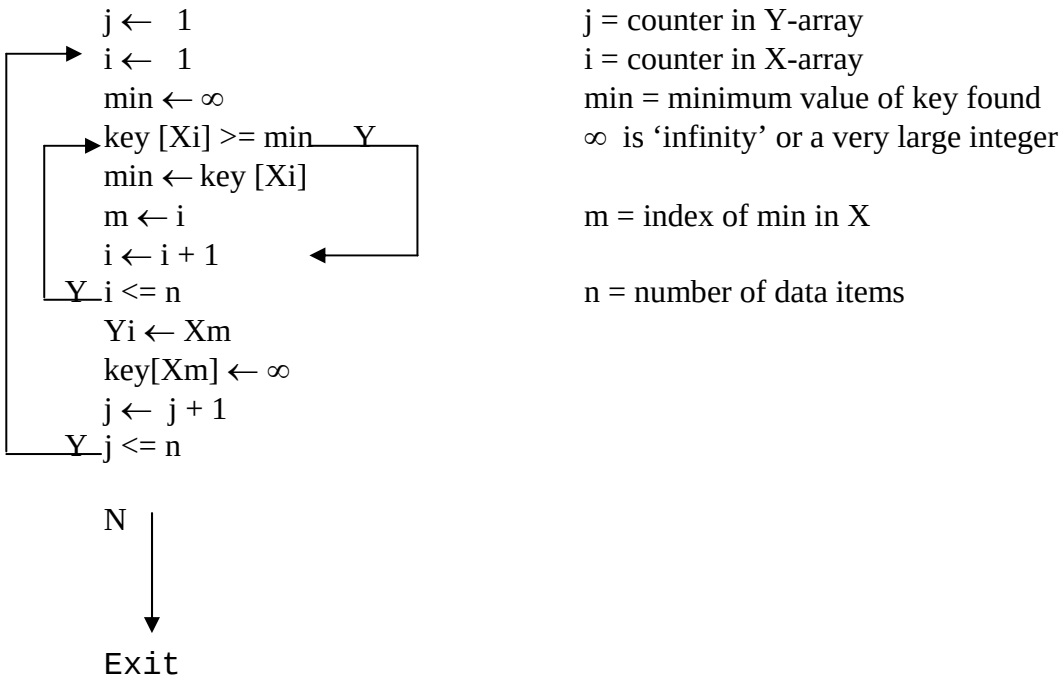
Copying out the question into the answer book gains no marks.

Very few students demonstrated any capability for algorithm development; code was always written without any preliminary sketches or descriptions. Consequently many answers had elementary mistakes. Thus no opening/closing files; leaving out 'READ' instructions or using record contents before reading it; many loops (usually 'FOR' loops) when one would suffice; always generating output even though the question did not ask for any, and above all the belief that syntactical correctness is more important than anything else. Many candidates rely on memorised code and have not developed the problem-solving ability that this area of work is meant to inculcate.

SECTION A

Question 1

An algorithm (published 1962 by K. Iverson) for the selection sort method is given below:



a) Translate this algorithm literally into a procedural programming language. (Thus the transfers of control implied by ⟶ must be translated into 'GO TO label' statements.

State which language you are using. **(10 marks)**

b) State where this algorithm could be useful in today's programming world.
Suggest why the algorithm is less useful now than when it was first available. **(6 marks)**

c) Modify the algorithm and translate it into PROCEDURE selectsort(…) which uses modern, structured code with meaningful variable names and without using labels. All external data values (input and output) must be passed to selectsort via parameters. **(14 marks)**

Answer Pointers

a) Iveson algorithm LITERAL translation

```
j = 1
label-1)  i = 1
min = large_value {system MAXINT or appropriate large integer}
label-2) IF key[ X[I]] >= min THEN GO TO label-3
min = key [X[I]]
n = i
label-3)  i = i + 1
IF i <= n THEN GO TO label-2
y[j] = X[m]
key [x[m]] = large_value
j = j + 1
IF j <= n THEN GO TO label-1
        ELSE GOTO exit
exit)
```

(10 marks)

b) This algorithm is still useful today as it has been thoroughly tested. So if one wants a quick implementation it can be translated as above with minimal testing. (**2 marks)**

It is useful in a low-level language which uses GOTO statements, simple 'IF' constructions and labels, and higher constructs like 'WHILE' are not available.

It is less useful in higher level languages, and is hard to understand as the variables have meaningless names which do not suggest their function. Names like 'i' and 'j' are particularly bad as the letters look alike, particularly on a poor-quality printout, which can lead to errors in coding.

There are no embedded comments; one would have to put them in. **(4 marks)**

c) It may be translated to PASCAL:

```
FOR i := 1 TO nitems
DO BEGIN
tem= := largevalue;
FOR j := 1 TO nitems DO
IF unslrted[j] is less than temp
        THEN BEGIN
        temp := unsorted[j]
        pos := j
        END;
sorted[i] := temp;

unsorted [pos] := largevalue
END
```

or COBOL version:

```
SELECTION-SORT SECTION.
TOP=LEVEL.
        MOVE ALL 9 TO LARGE-VALUE.
        PERFORM SCAN VARYING I FROM 1 BY 1
                UNTIL I GREATER THAN NITEMS
        EXIT-ROUTE.
                EXIT.
SCAN.
        MOVE LARGE-VALUE TO TEMP.
        PERFORM INNER-LOOP  VARYING J
FROM 1 BY 1  UNTIL J GREATER THAN NITEMS.
        MOVE TEMP TO SORTED(I).
        MOVE LARGE-VALUE TO UNSORTED(POS).
INNER-LOOP.
        IF UNSORTED (J) LESS THAN TEMP
                MOVE UNSORTED(J) TO TEMP
                MOVE J TO POS
        END-IF.
```

(14 marks)

Alternative encoding to equivalent standard accepted.

Examiner's Guidance Notes

An unpopular question. The discussion parts should not include vague statements. Precise and succinct answers were recognised as intelligent and plausible. In particular the examiner did not want a vague discussion about the purpose of algorithms in general, but where this one might be useful [e.g. low level programming] and poor features [e.g. single letter names, implied GO TO statements].

Candidates who know 'C' continue to include system instructions such as
 # include (**iostream.h)** and **# include (stdio.h)** when these do not answer the question, but are necessary for programs actually to run on a machine. This wastes their time, and suggests that the candidates do not know what the instructions do.

Question 2

Data concerning professional musicians is held on computer files having the following information:

family name	20 characters
address	4 lines each of 20 characters
instrument	20 characters
section	20 characters (e.g. string, woodwind)
capability	single-digit integer
appearance fee	pounds and pence
available	true or false
age	positive integer between 16 and 70.

a) Declare a record structure named "musician" to hold this information. **(4 marks)**

b) Specify variable declarations for:
   i) An array of these records called "player" capable of holding 100 items. **(2 marks)**
   ii) A serial file called "londoners" which holds many such records. **(2 marks)**

c) Write a program fragment and appropriate extra variable declarations to read the "londoners" file and selects from it 100 string players who are available and aged under 25 to form a youth orchestra. **(14 marks)**

d) Write another code fragment that reads the selection from (c) and sorts them into ascending order of appearance fee and prints out the family name, address and instrument for each one. **(8 marks)**

Answer Pointers

(a) TYPE string = PACKED ARRAY [1..20] OF CHAR;

```
    Musician = RECORD
    surname : string;
    address: ARRAY[1..4] OF string;
    instrument : string;
    capability : SHORTINT {INTEGER on other systems}
    fee: REAL
    available : BOOLEAN;
    age : 16.. 70   {SHORTINT if subrange facility not available}
        END;
```

(4 marks)

(b) VAR player : ARRAY [1..100] OF musician; onerec: musician;
londoners : FILE OF musician; **(4 marks)**

(c) {form youth orchestra}

```
RESET (londoners); ct := 0;
WHILE (NOT EOF(londoners)) AND )ct < 101) DO
    BEGIN
    READ(londoners, onerec);
    WITH onerec DO
        IF (section = string) AND (age < 30) AND aavailable
                THEN BEGIN
                ct := ct + 1;
                player[ct] := onerec
                        END
    END {while}
```

(14 marks)

(d) simple sort routine… others acceptable

```
VAR outer, inner: INTEGER; player : musician;
```

```
FOR outer := 1 TO (tot-1) DO
        FOR inner := (outer+1) TO total  DO
IF  player[outer].fee > player[inner]fee THEN BEGIN {swap outer and inner players}
        temp := player[outer];
        player[outer] := player[inner];
        player [inner] := temp
        END {if}
```
(8 marks)

Examiner's Guidance Notes

A popular question and the full range of marks awarded. These varied from nearly perfect answers to a majority who only knew how to make the declarations asked in parts (a) and (b), so denying themselves 22 of the 30 marks available.

Some algorithmic development in part (c) was desirable particularly for those who used C with its complex constructs. The best answers were from those who had used Pascal; some of those who used C were confused about its use of pointers, which were not needed here.

A small number of candidates used a database language but had little idea of what it actually did with the tables inside it.

Part (d) was largely memorised code of sort routines that candidates had experienced. Many of these wasted time on both input and output sequences, when the question did not ask for them. One candidate wrote six pages, half of which provided output routines for (c) and (d) which were not asked for. Another wrote four pages with input-checking and a complex structure. Lots of time must have been wasted here. The examiner does not penalise these extras – the candidate penalises themselves with the wasted time and lost opportunity for earning marks Another group wrote 'SORT' (usually without parameters) to answer (d), usually after long sequences requesting interactive input.

Question 3

a) Write a function, in Pseudocode, or Structured English or a programming language with which you are familiar, that implements searching as follows:

The function should accept an integer parameter ITEM that is the thing to be searched for.
The function should scan an array (name = LOCATED, size = LEN, both given as global in the context of the function).
The result of the function should be the index position in LOCATED if ITEM is found, or zero if ITEM is not found. **(18 marks)**

b) Dry-run your pseudocode from (a) above with the following data:

ITEM = 16
LEN = 6
LOCATED =

3
5
7
9
11
13

(12 marks)

Answer Pointers

a) The model algorithm should exhibit the following:

- Accept as parameter the ITEM, and prepare to deliver the index position in LOCATED
- Initialise local variables including some flag or marker that ITEM has been found, initially 'null'.
- Set up a single loop to scan LOCATED, clearly predicated at end on some test of whether the ITEM has been found in LOCATED
- Control of this loop is determined by LEN and an appropriate test with the value of the array index.
- Procedure terminates when ITEM is found or when max size of LOCATED is reached, remembering to test last index in LOCATED.

Three marks for correct implementation of each step. Total 18 marks.

b) The model 'dry run' is:

Flag	ITEM	Loop	LEN	LOCATED	Flag setting
Preset NULL	16	1	6	3	No
		2		5	No
		3		7	No
		4		9	No
		5		11	No
		6		13	LEN reached
Null		6		13	
	Stops. Flag still null			NOT FOUND return ZERO	

The same 5 steps should be present, two marks each. Total 12 marks.

Examiner's Guidance Notes

a) Some candidates designed an algorithm that recognised ITEM but proceeded to scan until the end of the array. Other, fewer, candidates forgot to initialise local variables or arrange input parameter handling, making results of later tests indeterminate. Others decided this was a binary search algorithm, because the data was ordered but despite being told in the question to write a scanning algorithm.

Most candidates used a 3GL-type language. Others used a form of database macro language but, mostly, the essence of the programming was still discernible.

b) The dry run was mostly very well done. Candidates mostly worked on a mental model of what they thought they had designed in (a). For some of the poor algorithms, it was clear that candidates had a clear mental picture, because the dry run was often more correct than the algorithm.

Some candidates got to the end but could not identify what was the result.

Question 4

a) Describe the principles of Modular Programming. Comment particularly on execution flow and the handling of parameter data. **(18 marks)**

b) State TWO benefits of using Modular Programming methods. Give your reasons. **(12 marks)**

Answer Pointers

a) A model description is that modular programming is about single-entry and single exit points of procedures or modules of functionally-focussed code. Parameters are declared at procedure head and should avoid use of global variables. Parameters may be typed for read-only or ability to be returned out of the procedure.

10 marks for description of control features. 5 marks for descriptions of parameter handling.

b) Expected benefits were two from the following list:
Productivity – can break problem into parts and factor over many programmers. Testability – because a module is functionally focussed it is easier to test. Global variables are obtained via parameter strings, and so inter-modular errors (confusions over global settings) are avoided. Maintainability – module construction eases localisation of error, understanding and repair. Reusability – self-contained modules can perform in a variety of contexts.

Marking – any two, similar and plausible, with reasons, 10 marks.

Examiner's Guidance Notes

a) Most candidates described control features well enough but did poorly on parameters. Some candidates confused Modular Programming with Structured Programming and gave answers that described sequence, selection and iteration.

b) Some candidates gave the same answer twice e.g. modules are easy to work with because of the tight, functional definitions; and modular development saves time because modular definitions are functionally straightforward and simple.

SECTION B

Question 5

a) An interactive program is to accept N real numbers and then evaluate and print the Arithmetic mean (A) and Geometric mean (G) defined by the following formulae:

$A = (R_1 + R_2 + \ldots + R_N) / N$

$G = (R_1 * R_2 * \ldots * R_N) \uparrow (1 / N)$ [$\uparrow$ means 'to the power of']

Develop the algorithm and write code for the program. State which programming language you have used. **(8 marks)**

b) Some procedural languages do not have a 'power' operator. Show how you would implement the calculation for 'G' in this situation (e.g. Pascal). **(4 marks)**

Answer Pointers

(a) **(8 marks)**

```
PRINT "MEANS PROGRAM"
atot = 0
gtot = 1
PRINT "How many real numbers to be input?"
INPUT N
FOR ct = 1 TO N
PRINT "Input item "; ct
INPUT R
atot = atot + R   {holds sum of all terms}
gtot = gtot * R   {holds product of all terms}
NEXT ct
A = atot / N
G = gtot ^ (1 / N)
PRINT "arithmetic mean = "; A
PRINT "geometric mean  = "; G
END
```

(b)
Multiplying terms may lead to 'overflow' especially if the calculations are done using INTEGER arithmetic. Other implementations can cope with large real numbers; some provide a facility for extending the standard range available.

$G = (R_1 * R_2 * \ldots * R_N) \uparrow (1 / N)$
$\log G = \log (R_1) + \log(R_2) + \ldots + \log(R_N) / N$

so form the sum of the logarithms of the R-terms, divide by N then use the antilogarithm function.

(4 marks)

[alternative (b)] It is possible to avoid this by judicious use of logarithms thus:

Using logarithms to any base:
$g = gtot \uparrow (1 / N)$
$\log g = \log (gtot) / N$
then $g = alog [\log (gtot) / N]$
where log and alog functions are provided, usually to base 10; base *e* can be used.

Examiner's Guidance Notes
Much less popular; and the full range of marks awarded. Most candidates produced no algorithm despite its being specifically asked for. Candidates had surplus 'READ' instructions, several 'FOR' loops and the like. Most assumed that ' ^' was the power operator in 'C' presumably from earlier use in basic.

The most serious error was to directly copy the formulae for 'A' and 'G' to the program including the '…' (ellipsis) used in the formula:

$A = (R_1 + R_2 + \ldots + R_N) / N$

$G = (R_1 * R_2 * \ldots * R_N) \uparrow (1 / N)$ [$\uparrow$ means 'to the power of']

This betrays a serious ignorance of programming, that instructions can be implied in this way.

Few candidates offered an answer to part (b). Marks were given for loops which formed the product, then some function like 'pwr' which did the n'th root. Very few indeed knew of any 'C' function which might be used such as 'pwr'. Some candidates again used the '^' operator despite the question specifically saying this was not available. Only one thought of using logarithmic functions, provided in languages like 'C' and PASCAL. Nearly all procedural languages provide a 'log' function.

Question 6

A palindrome can be described as an array of characters in which the first and last letters are the same and those characters between them also form a palindrome. Examples are PEEP, RADAR.

Write a pseudocode function '**palcheck**' which takes as its parameter a character array and which returns 'TRUE' if the array contains a palindrome, otherwise returns 'FALSE'. You may use either an iterative or recursive method.

(**algorithm 6 marks)**
(code 6 marks)

Answer Pointers

(Iterative method to check for a palindrome) PASCAL implementation

```
FUNCTION palck ( letter: ARRAY to hold palindrome; length:of palindrome)
fwd = 1;  bwd = length;   finished := FALSE;
WHILE bwd > fwd) AND (NOT finished) DO
        BEGIN
        IF letter[fwd] = letter[bwd]  THEN
                BEGIN
                fwd = fwd + 1
                bwd = bwd - 1
                END
        ELSE
                BEGIN
                WRITE "not a palindrome"; finished := TRUE
                END
        END { palindrome check}
```

(4 marks)

Essentials of runnable program:

```
CONST maxlength = 20;
TYPE chtype = PACKED ARRAY[1..maxlength] OF CHAR;
VAR inword:chtype; kt,tot:INTEGER;

FUNCTION palck(letter:chtype; ct,length:INTEGER):BOOLEAN;
VAR finished:BOOLEAN; fwd, bwd:INTEGER;
BEGIN
fwd = 1;  bwd = length;   finished := FALSE;
WHILE (bwd > fwd) AND (NOT finished) DO
      BEGIN
      IF letter[fwd] = letter[bwd]  THEN
            BEGIN
            fwd = fwd + 1
            bwd = bwd - 1
            END
      ELSE
            BEGIN
             finished := TRUE
            END
      END { palindrome check}
```

(4 marks)

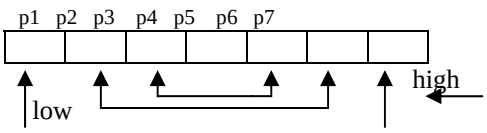
```
BEGIN  {top level}
```

```
WRITELN('input word for testing if a palindrome');
tot := 0;
WHILE NOT EOLN DO
   BEGIN
   tot := tot + 1;
   READ(inword[tot]);
   END;
WRITELN('length of input string = ',tot:2);
WRITELN(inword);
IF palck(inword,tot) THEN WRITELN('is a palindrome')
   ELSE WRITELN('is NOT a palindrome')
END.                                                    (4 marks)
```

Marks awarded in equal proportion for algorithm and code.

Diagrams like that given below were also accepted as an algorithm, appropriate marks being awarded for greater detail:

ARRAY representation



Comparisons needed:
p1 with p7, p2 with p6, p3 with p5

A redundant comparison of p4 with itself was not considered an error.
This diagrammatic beginning is much better than function declarations, counters set to zero and so on as it suggests what is also needed, e.g. the length of the palindrome (often wrongly set to a constant.)

Examiner's Guidance Notes

This question was quite popular as many had seen this problem worked through, so it was generally attempted from memory. The full range of marks was used. Answers varied from the completely correct to those who could recall very little and so wrote a vague algorithm. Most did the complete task inside the procedure (although a function was asked for) including input of the test word and display of the results.

Marks were awarded for those who wrote a complete program to check if a given word was a palindrome or not, although the question did ask only for the palindrome part, both algorithm and code. Many candidates clearly do not know what belongs in an algorithm and what should be left until the program:

No extra marks were gained for identical material under 'algorithm' and 'program'. Added value was what was expected. Candidates are most reluctant to show any development; if there was any this was invariably crossed out as 'rough work'.

This diagrammatic beginning is much better than function declarations, counters set to zero and so on as it suggests what is also needed, e.g. the length of the palindrome (often wrongly set to a constant.)

In fact interpretations of just what should go into such a function varied. Some included data input, Many more confused 'function' with 'procedure'. This was not penalised, especially if candidates had used 'C'.

Candidates should also consider the appropriateness of what they are using, particularly if this is memorised code. One answer had a complete linked-list builder with 'push' and 'pop' procedures to store the palindrome, then a means of retrieving the data so stored. It ran to three full pages of code, which must have taken a lot of time to write. It was also obvious that some candidates wrote the code

first, then abbreviated it a little to become the algorithm. It is only when presented with a problem of some length and complexity that the need for an algorithm becomes obvious to candidates.

Question 7

An ordered linked list has been set up with one *data item* and one pointer only in each *element* of the list.

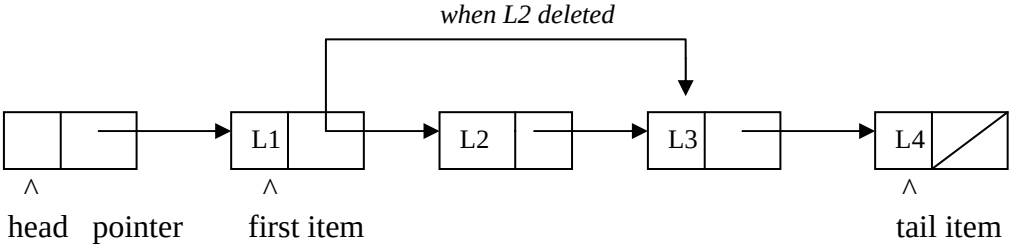
a) Draw a diagram of this linked list (with five *elements*) with a pointer to its head. Show the pointer movements when second *element* of the list is deleted. **(3 marks)**

b) Write a program declaration of the data structure needed. **(2 marks)**

c) Write pseudocode for a procedure 'deleteitem', with appropriate parameters, to search for the *data item* held in the variable 'rejectitem' and delete the *element* holding that *data item* from the linked list. You can assume that an *element* holding the *data item* exists in the linked list. **(7 marks)**

Answer Pointers

(a)



(3 marks)

(b)

```
TYPE        ptr    = ^node;
            node = RECORD
            data : (appropriate data type);
            next : ptr
                END;
```

(2 marks)

(c)

Essential idea:

search for data item by list traversal

IF list item = item for deletion THEN effect pointer movement

```
PROCEDURE deleteitem ( head: ptr; rejectitem: (appropriate data type) );
traverse : ptr;   ; done : Boolean;
BEGIN
traverse := head;
done := FALSE;
WHILE (traverse^.next <> NIL)  AND  not done   DO
BEGIN
IF    traaverse.data = reject item THEN
                                    BEGIN
                                    traverse.next := traverse.next.next
                                    done := TRUE
                                    END
                                ELSE     traaverse := traverse^next;
```

END {while loop}
END; {procedure deleteitem} **(7 marks)**

Examiner's Guidance Notes
A popular question with the full range of marks used. This was a bookwork question, with clear limits established on what could be asked. It was answered almost entirely with memorised code, which sometimes resulted in students coping down the wrong area that they had memorised. But generally it was done well by anyone who had studied this area of the syllabus. Many used it as a last question as they knew the deletion pointer movements and the declaration. However, nobody checked that what was written would work.

Question 8
You are asked to design the testing strategy for a spreadsheet that contains both simple data and text, and computed data.

a) Give TWO examples of 'White Box' tests you would use**.** **(6 marks)**
b) Give TWO examples of 'Black Box' tests you would use. (**6 marks)**

Answer Pointers
a) Examples of **White box tests on a spreadsheet** are testing for correct logic e.g.
macros, derived or calculated fields, checking formulas and areas of spreadsheet corresponding to pictures to be drawn.

Marking was 6 marks, clearly 3 marks for each example

b) Examples of **Black box tests on a spreadsheet** are checking functions such as diagrams, pie charts against prepared answers from a desk calculation,
and generally to check answers against input data separately checked with a hand calculator or similar.

Marking was 6 marks, clearly 3 marks for each example

Examiner's Guidance Notes
Many candidates ignored the context of a spreadsheet and gave generic and descriptive answers. Many other candidates gave no context at all, merely descriptions of what the names White and Black mean in testing.

Each part asked for two examples. Despite this, some candidates only supplied one example in each part.

Other candidates said they would white-box test such things as addition in the spreadsheet. These answers were not acceptable, because this amounted to white-box testing the software that implemented the spreadsheet rather than the application programmed into the spreadsheet.

Question 9

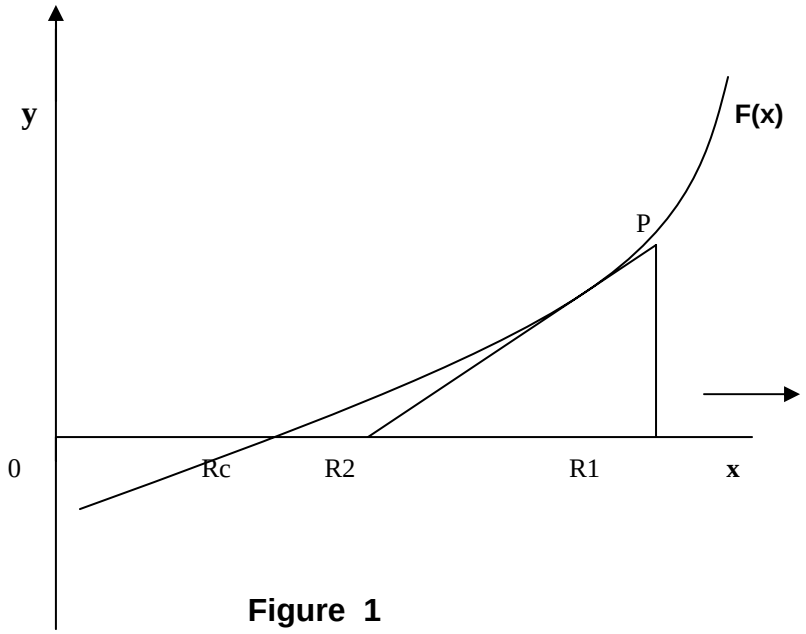


Figure 1

Figure 1 shows a graph of a function y = F(x) which has a real root at x = Rc.

If R1 is a first approximation to this it can be shown that R2 is a better approximation where

$$R2 = R1 - \frac{F(R1)}{F'(R1)}$$

Newton's formula
F'(x) being the first derivative of the function F(x).

a) Develop an algorithm to find the three roots of the equation where

$F(x) = Ax^3 + Bx^2 + Cx + D$ and
$F'(x) = 3Ax^2 + 2Bx + C$

The approximate values of the roots (root1, root2 and root3) are to be input interactively. The algorithm should terminate when the positive difference between two successive root calculations is less than or equal to 0.001. **(6 marks)**

b) Write a FUNCTION '**newton**' with appropriate parameters to apply this method. **(6 marks)**

Answer Pointers

(a) *Develop an algorithm to find the roots of the equation where*
$F(x) = Ax^3 + ^{Bx2} + Cx + D$ and $F'(X) = 3Ax^2 + 2Bx + C$ the approximate values of the roots being root1, root2 and root3 which are input interactively. **(6 marks)**

Algorithm for roots by Newton's Method:
DEFINE F(x) as the function of x, here Ax3 + Bx2 + Cx + D
DEFINE FD(x) as the derivative function here 3AX2 + 2Bx + C
LOOP 3 times
INPUT approx. root value R1
INPUT end condition or value by which successive roots are not to differ E
Ct=0
Label) R2 = R1 - F(R1) / FD(R1)

```
IF ABS(R2 - R1) GREATER THAN E THEN
        Increment ct
        R1 = R2
        GO TO label
PRINT final root = R2   Iterations = ct
END
```
This needs to be repeated for every root value.

(b) *Write a FUNCTION 'newton' with appropriate parameters to apply this method.* **(6 marks)**

Implementation in Qbasic:

```
DIMENSION root[3]
PRINT "roots of cubic equation by Newton's Method"
PRINT "Input equation coefficients A,B,C,D from equation Ax^3 + Bx^2 + Cx + D"
INPUT A,B,C,D
PRINT "Input successive root values"
PRINT "input end condition or difference between successive roots to finish"
INPUT  E

DEFINE FUNCTION F(x) = A*x^3 + B*x^2 + C*x + D
DEFINE FUNCTION FD(x) = 3*A*x^2 + 2*B*x + C

FOR ct1 = 1 TO 3 DO
PRINT "root "; ct1;
INPUT root[ct1]
ct2 = 0
Label: R2 = R1 - F(root[ct])  /  FD(root[ct])
IF ABS(R2 - R1) > E THEN
ct2 = ct2 +1
R1 = R2
GO TO Label
ENDIF
PRINT "final root = "; R2; " iterations needed"; ct2
END
END FOR

NEXT ct1
```

Examiner's Guidance Notes

A popular question with the full range of marks used. This was a bookwork question, with clear limits established on what could be asked. It was answered almost entirely with memorised code, which sometimes resulted in candidates' copying down the wrong area that they had memorised. But generally it was done well by anyone who had studied this area of the syllabus. Many used it as a last question as they knew the deletion pointer movements and the declaration. However, nobody checked that what they had written would work.

Question 10

Describe the operation of the index to a file. Illustrate your description with suitable diagrams, and include how the index is maintained as the file is updated. **(12 marks)**

Answer Pointers

The answer was expected to show a diagram with pointers from index to file locations. In addition, the answer should show an understanding of how the index grows and shrinks to follow the file insertions and deletions.

4 marks for basic understanding of an index, 4 marks for the picture, 4 marks for the discussion of growth and shrinkage.

Examiner's Guidance Notes

Many candidates managed the basic understanding and/or the picture. Few described both. Fewer still could describe index growth and shrinkage. It was noteworthy that candidates had no model of how the points were earned.

The total of 12 marks is reasonably analysed as four 3s or three 4s. That is to say, the answer should deliver three or four distinct parts. Very many candidates gave a one-part answer.

Question 11

Describe TWO Web-site GUI used for each of the following actions:

a) Selection of items in a browser. **(6 marks)**

b) Keeping navigation back to the primary webpage always visible when clients click-though to go to other parts of the website. **(6 marks)**

Answer Pointers

a) Selection of items in a browser. Radio buttons, drop-down lists, clickable icons, clickable hyperlinks. Any two or plausible alternative. 3 marks each

b) Keeping navigation back to the primary web page always visible when clients click-though to go to other parts of the website. 'Frames' approach to maintain site structure navigation, or opening a link in a new page smaller in front of home page always visible behind. Or plausible alternative. 3 marks each area

Examiner's Guidance Notes

Many candidates again failed to supply the requested two examples from each section. In (b), many candidates did not comprehend the requirement to keep the primary page visible. Answers that said, in effect, 'use the back button on the control bar' were not acceptable. The question sought knowledge of styles of web design to achieve certain results. Even an answer that said 'create a back-button as part of the page' received some merit.

Some candidates described the functions of a web browser, but did not give any mechanism by which they could be achieved; for example, colours and size are important, but no description of a 'clickable icon'. Similarly, speed of loading is important, and a GUI should perhaps limit the number of pictures it contains, but what has that to do with GUI elements defined to aid navigation?

Question 12

Using suitable diagrams to illustrate your answer, briefly describe the operation of the following elements of system software:

a) The scheduler **(6 marks)**
b) The compiler **(6 marks)**

Answer Pointers

a) The scheduler; this question expected the answer to be a picture of a queue and description of a priority scheduler, or picture of a list and description for a round-robin or pre-emptive scheduler. Picture earned 3 marks, and description 3 marks.

b) The compiler; the question expected the answer to be a picture or a block diagram and description of compiler phases such as lexical scan and code generation. Picture 2 marks. Description 4 marks.

Examiner's Guidance Notes

Some candidates cited the event scheduler in application software such as MS Outlook. Such answers received some merit. Few candidates scored well in both sections (a) and (b). This was surprising.