BCS HIGHER EDUCATION QUALIFICATIONS
BCS Level 4 Certificate in IT

October 2010

EXAMINERS' REPORT

Software Development

General

Avoid whole pages of crossing out - it wastes an inordinate amount of time.

Do not copy the question into the answer book. This gains no marks and wastes time.

If a declaration of records/pointer structure is developed in an early part of a question, it need not be copied into later parts of the answer but an appropriate place for them should be indicated.

System commands associated with 'C' [uses crt, clrscr, #include <iostream.h>] are not needed.

Avoid writing answers in the wrong answer book. This is a nuisance as they have to be marked out of sequence with the majority of answers.

Section A

Question A1

Write an algorithm to implement the following word game for 2 players:

Each of the players in turn enters a word where all the letters are different, starting with 4 letters. The word is then checked to see if it has any letters the same; if it has that player is eliminated. There must be no duplicated words. The next round has words with 5 letters. This continues until only one player remains; he/she is declared the winner.

a) Specify the following record named '**oneplayer**' in a language of your choice, to use in the program. State which language you are using.

(3 marks)

name, word (10 characters)
Playerout _indicator (in, eliminated)

b) Write a procedure **checkword** with appropriate parameters, which checks if the letters in 'theword' are all different. If they are it returns FALSE otherwise it returns TRUE.

(6 marks)

c) Develop an algorithm to implement the word game. It must show at least two stages of development, and reach a stage where coding would be straightforward.

(Algorithm 11 marks)
(Development 10 marks)

Answer Pointers

An indication of the level of detail required is given below

```
oneplayer = RECORD
                name, word : string;
                Playerout : BOOLEAN
            END;

PROCEDURE checkword(theword:shortstring; VAR result:BOOLEAN);
    VAR act,thelen:INTEGER;  letset:SET OF CHAR;
BEGIN
    thelen := LENGTH(theword);
    result := TRUE; act := 0;letset := EMPTY;
    WHILE result  AND (act < thelen) DO
          BEGIN
             ADD 1 to act
             (*check if letter already in set.  IF letter already there, return FALSE. Otherwise add letter to
             set*)
             IF theword[act] IN letset THEN
                 result := FALSE
             ELSE letset := letset + theword[act];
          END
END;
```

Initial algorithm

```
INPUT howmany
FOR ct = 1 TO howmany DO
      INPUT players name
wordlen = 3
Leftin = howmany
WHILE leftin IS GREATER THAN 1 DO
      ADD 1 TO wordlen
      FOR pos = 1 TO leftin DO
            PRINT "input word of length" wordlen "for player" name
            INPUT word
            checkword(word, different)
            IF NOT different THEN
                  PRINT "player" name "eliminated"
                  Playerout = TRUE
                  SUBTRACT 1 FROM leftin
PRINT winner's name
```

Development

```
PROGRAM wordgame(INPUT,OUTPUT);
TYPE      shortstring=string[10];
          oneplayer=RECORD
          record  as defined earlier

VAR       aplayer:ARRAY[1..5] OF oneplayer;
          different:BOOLEAN;
          wordlen,howmany,leftin,ct,pos:INTEGER;

code for procedure checkword here
BEGIN  {TOP LEVEL}
    WRITELN('word game Summer 2007');
    WRITELN('INPUT howmany players ');
    READLN(howmany);
    WRITELN('input players names');
    FOR ct := 1 TO howmany DO
        BEGIN
          WRITE('INPUT name of player ',ct:2,' ':2);
          READLN(aplayer[ct].name)
        END;
    wordlen := 3;
    leftin := howmany;
    WHILE leftin > 1 DO
    BEGIN
          wordlen := wordlen + 1;
          FOR pos := 1 TO leftin DO
          BEGIN
              WITH aplayer[pos]  DO
            BEGIN
                WRITELN(' input word with length ',wordlen:2,' for player ',name);
                READLN(word);
                chekword(word,different);
                    IF NOT different THEN
                        BEGIN
                          WRITELN('player ',name,' eliminated');
                          playerout := TRUE;
                          leftin := leftin - 1
                        END
            END
          END
    END;
(* print winning player's name *)
    FOR ct := 1 TO howmany DO
    IF NOT aplayer[ct].playerout THEN WRITELN(aplayer[ct].name,' is the WINNER');
    WRITELN('PROGRAM ENDS')
END.
```

Examiner's Guidance Notes
Answers to this question were few and far between. Nearly all were poorly done and worth only single-figure marks. It is foolish to attempt an 'A' section question and know only the record structure which in this case was worth 3 marks.

Some candidates used flowcharts which have been superseded as a design methodology and are quite unsuitable here. Likewise the tabular method is unsuitable (e.g. it does not deal with files well)

Input	Processing	Output
INPUT(variables)	calculations	Desired output

Instructions such as #include <stdio.h> or #include <conio.h> have no place in a question on algorithmic development. One answer had definitions of algorithm, iteration and GUI but no use of any of them.

Question A2

The following code uses Newton's Method to improve the roots of a quadratic equation

$$Ax^2 + Bx + C = 0$$

The roots are approximately known.

Line No.	Code	Version B

10
11
12
13
14
15
16
17
18
19
20
21
	FNX(X) = A*X↑2 + B*X + C
	FND(X) = 2*A*X + B
	PRINT "EXECUTION
	STARTS HERE"
	K = 0
	R2 = R1 - FNX(R1) /
	FND(R1)
	K = K + 1
	PRINT "count = ",K," root = ",
	R2
	IF K = 2 THEN GO TO 20
	R1 = R2
	GO TO 15
	WRITE(" final root = ", R2)
	END
		float FNX(X){
		return(A*X*X+B*X+C);}
		float FND(X){return(2*A*X+B);}
		printf("EXECUTION STARTS
		HERE");
		K = 0;
		loop: R2 = R1 - FNX(R1) /
		FND(R1);
		K = K + 1;
		printf("count = %d root = %f", K,
		R2);
		if(K == 2) goto end;
		R1 = R2;
		goto loop;
		end: printf("final root = %f", R2);
		}

a) Dry run this code with the values A = 1, B = -3, C = -10 and R1 = 6. Give two cycles of the loop which show that the root R1 is converging to a value of 2.6.
(20 marks)

b) Re-write the code as a function with parameters A, B and C and which reads in the first value of R1.
Use better names for the variables and avoid the use of 'GO TO' statements.
(10 marks)

Answer Pointers

Marks were awarded for the variables in the table which have changing values. Thus the equation has constants A, B C which do not need columns. [No penalty was made if these were given columns.]

The necessary ones were

Line number	Instruction	K	R1	R2	IF…Then true/false	Output
1 mark….	……..	1 mark	1 mark…	……..	1 mark	1 mark

Line	Instruction	K	R1	R2	FNX()	FND()	IF	Output/notes
	PRINT	?	6	?	?	?		Execution starts here
	Assign	0						
				5.1	8.0	9.0		R2=6.0 – 8/9 = 5.1
	Assign	1						
	PRINT							count 1 root = 5.1
	IF						False	
	Assign		5.1					
	GO TO loop							
Loop	Assign			5.0	0.7	7.1		
	Assign	2						
	PRINT							Count 2 root = 5.0
	Assign		5.0					
	IF	2					true	GO TO end
End	PRINT							Final root = 5.0
	END							

```
WRITELN('execution starts here');
WRITELN('INPUT approx. value of root');
READ(old_root);
X := old_root;
WRITELN('INPUT the coefficients A, B, C of the quadratic equation Ax^2 + Bx + C = 0');
READ(A, B, C);
FUNCT_X := A*X*X + B*X + C
DERIV_X := 2*A*X + B
Error := 1.0;
Count := 0;
WHILE (Error > 0.0005 DO
    BEGIN
        New_root := Old_root - FUNCTX / DERIV_X;
        Count := Count + 1;
        WRITELN( "count = ",Count," root = ", New_root;
        Error = ABS(New_root - Old_root);
        Old_root : = New_root
    END;
```

```
WRITELN(" final root = ", New_root)
END.
```

The expressions FUNCTX := A*X*X + B*X + C can be expressed in functional form thus:

```
FUNCTION FUNCTX (A, B, C, X : REAL) : REAL:
BEGIN
   FUNCTX := A*X*X + B*X +C
END;
```

thus:

```
FUNCTION DERIV_X (A, B, X : REAL) : REAL:
BEGIN
   DERIV_X  :=  2*A*X  +  B
END;
```

Examiner's Guidance Notes

This was a popular question; the full range of marks was used. Too many candidates just copied out the given code, with no values for the dry run or modifications for part (b). Others got some way into the dry run table, then crossed it all out, sometimes as much as two pages of work. This would have wasted a lot of the candidates' time. Others did not understand the use of R1 as parameters for functions FNX(X) and FND(X) and consequently could get not values for R2.
There is not time to work out the dry run in rough, then make a fair-copy version. Candidates need to practice such answers without extensive rough work.

The worst mistake in part (b) was not to realize that the GO TO …label statements had to be replaced by a WHILE or FOR loop. Very few candidates altered any variable names. It is important that code is intelligible to other programmers; long gone are the days when only single letters of the alphabet could be used.

Question A3

The arrays **x, y** have been initialised as follows

index	0	1	2	3	4	5	6	7	8	9
x	41	19	55	90	80	76	13	55	1	0
y	42	20	57	90	81	76	12	49	0	1
z										

The subroutine ***r*** in the code below is going to be executed with parameter ***s*** set to 2 and parameter ***t*** set to 7. [*You can choose to follow either version of the code*]

a) Trace the call of the subroutine ***r(2,7)*** and show clearly the results of the call.

(8 marks)

	Version A	Version B
1	void r(int s, int t){	PROCEDURE r(s, t : INTEGER);
2	int v, w;	VAR v, w : INTEGER;
3	/* begin function */	BEGIN
4	for(w=s; w<=t; w++)	FOR w := s TO t DO
5	{	BEGIN
6	v = x[w] + y[w];	v := x[w] + y[w];
7	z[w] = v / 2;	z[w] := v / 2;
8	}	END
9	}	END;

b) Write a brief summary of what the subroutine does.

(6 marks)

c) Decide on better names for the identifiers (the subroutine name, its parameters and the variables) and rewrite the code [either version A or version B] using your new names and including suitable comments.

(10 marks)

d) Rewrite lines 4 to 8 [of either version A or version B] using a while-loop instead of a for-loop.

(6 marks)

Answer Pointers

a)

	Version A	Version B
1	r(2,7)=	r(2,7)=
2	{	VAR v, w : INTEGER;
3	int v, w;	BEGIN
4	for(w=2; w<=7; w++)	FOR w := 2 TO 7 DO
5	{	BEGIN
6	v = x[w] - y[w];	v := x[w] – y[w];
7	z[w] = (x[w] + y[w]) / 2;	z[w] := (x[w] + y[w]) / 2;
8	}	END
9	}	END;

index	0	1	2	3	4	5	6	7	8	9
x	41	19	55	90	80	76	13	55	1	0
y	42	20	57	90	81	76	12	49	0	1
z	NB still blank	NB still blank	56	90	80.5	76	12.5	52	NB still blank	NB still blank

2 marks for handling the parameters
2 marks for handling the trace of the loop
4 marks for obtaining final result for r(2,7) is shown in changes of z array

b) value of element in z is average of corresponding array elements of x & y
for elements with subscripts s to t

c)
r -> aveZ
s -> lower
t -> upper
v -> sum
w -> index

	Version A	Version B
1	void aveZ(int lower, upper)	PROCEDURE aveZ(lower, upper : INTEGER)
2	{	VAR sum, index: INTEGER;
3	int sum, index;	BEGIN
4	for(index=lower;index <=upper;index++)	FOR index := lower TO upper DO
5	{	BEGIN
6	sum = x[index] + y[index];	sum := x[index] + y[index];
7	z[index] = sum / 2;	z[index] := sum / 2;
8	}	END
9	}	END;

d) Rewriting lines 4-8 using a while loop

	Version A	Version B
1		
2		
3		
4	index = lower;	index := lower;
5	while(index <= upper)	WHILE index <= upper DO
6	{	BEGIN
7	sum = x[index] + y[index];	sum := x[index] + y[index];
8	z[index] = sum / 2;	z[index] := sum / 2;
9	index++;	index := index + 1
10	}	END;

Examiner's Guidance Notes
This was a popular question.

a) Getting the 'right answer' is only worth half the marks. Some candidates did not trace the call of the subroutine and they just provided some part of the 'right answer'. A few candidates had for example x[55], y[57] in their traces, showing a confusion between the subscript/index of an array and the contents at that subscript/index.

b) There are six points to be made. "The function r uses the subscripts **s** and **t** and calculates the value of each element in **z** which is the **average** of corresponding array elements of **x** & **y.**"

The function does NOT sort the values in the array z.
The function does NOT find the maximum value in the array z.

c) When choosing new names some candidates only gave a new name for the function, or for the function and parameters, but not local variables. Some of the new names given were no better than the original (e.g. a, b, m, n, k, l). Comments were often absent or several paragraphs of separated text. What was required was in-program comments. Most of the comments are superficial like 'start of function', 'start of loop', 'end of loop', 'end of function'.

d) This part was mostly satisfactory. Some candidates merely changed the word 'for' to 'while', showing a confusion between these loop statements. A few candidates missed the initialisation of the index (e.g. index=0). Many others put a reasonable condition after the while, but then forgot to add in the counting code (e.g. index++). Those that included the counting often put it in the wrong places (e.g. index++ outside the loop).

Question A4

program A	program B

/* program A */
int res;
int convert(char c){
  int val;
  /* begin function */
  val = 0;
  if(c=='l' || c=='i')
  val = 1;
  else if (c=='V' || c== 'v')
  val = 5;
  else if (c=='X' || c=='x')
  val=10;
  return(val);
}
void main(){
  res = convert('X') +
convert('i');
}
	PROGRAM B;
	VAR res : INTEGER;
	FUNCTION convert(c : CHAR) :
	INTEGER;
	VAR val : INTEGER;
	BEGIN
	  val := 0;
	  IF (c='l') OR (c='i') THEN
	  val := 1
	  ELSE IF (c='V') OR (c= 'v') THEN
	  val := 5
	  ELSE IF (c='X') OR (c='x') THEN
	  val :=10;
	  convert := val;
	END;
	BEGIN
	  res := convert('X') + convert('i')
	END.

a) Choose either program A or program B and then find and copy out an example of each of the following.
[*Take care to copy out only what is requested, nothing more*]

(1 mark each, 12 total)

a.1) a reserved word a.2) a variable identifier a.3) a type identifier a.4) a function identifier a.5) a character constant a.6) an integer constant	a.7) an arithmetic operator a.8) a logical operator a.9) a formal parameter a.10) an actual parameter a.11) a local variable a.12) an assignment symbol

b) Continuing with either program A or program B as in part (a), find and copy out an example of each of the following.
[*Take care to copy out only what is requested, nothing more*]

(3 marks each, 18 total)

b.1) a function call b.2) a function declaration b.3) a variable declaration	b.4) an assignment statement b.5) an expression with a boolean (logical) value b.6) a conditional statement

Answer Pointers

(Note: candidates were only requested to give one example, though the table below offers multiple examples)

	program A	program B
a.1	if, else	PROGRAM, FUNCTION, BEGIN, END, IF, THEN, ELSE, VAR
a.2	val, res	val, res
a.3	int, char, void	INTEGER, CHAR
a.4	convert	convert, main
a.5	'I', 'i', 'V', 'v', 'X', 'x'	'I', 'i', 'V', 'v', 'X', 'x'
a.6	0, 1, 5, 10	0, 1, 5, 10
a.7	+	+
a.8	==, ||	=, OR
a.9	c	C
a.10	'X', 'i'	'X', 'i'
a.11	val	val
a.12	=	:=

b.1	convert('X')	convert('X')

b.2	int convert(char c)
	{
	  int val;
	  val = 0;
	  if(c=='I' || c=='i')
	  val = 1;
	  else if (c=='V' || c== 'v')
	  val = 5;
	  else if (c=='X' || c=='x')
	  val=10;
	  return(val);
	}
		INTEGER FUNCTION convert(c:CHAR);
		VAR val : INTEGER;
		BEGIN
		  val := 0;
		  IF (c='I') OR (c='i') THEN
		  val := 1
		  ELSE IF (c='V') OR (c= 'v') THEN
		  val := 5
		  ELSE IF (c='X') OR (c='x') THEN
		  val :=10;
		  convert := val;
		END;

b.3	int res; int val;	VAR res : INT; VAR val:INTEGER;

b.4	val = 0;
	val = 1;
	val = 5;
	val = 10;
		val := 0;
		val := 1;
		val := 5;
		val := 10;

b.5	c=='I' and c=='i', etc	c='I' and c='i', etc

b.6	if (c=='X' || c=='x')
	  val=10;
		IF (c='X') OR (c='x') THEN
		  val :=10;

Examiner's Guidance Notes
There were far too many poor answers.

(a) Wrong answers were mostly too long - e.g. " ELSE IF (c='V') OR (c= 'v') THEN" might be written when "=" was the answer.

(b) return(...) is not a "function call". "Function call" and "function declaration" were muddled. Many answers to b.2 only had the first line. Some answers to b.6 were spoilt by including return(val) or convert:=val as part of the conditional. convert(...)+convert(...) is not a function call - it is an arithmetic expression involving 2 function calls.

SECTION B

Question B5

A serial file 'datafile' has a sequence of records R_1, R_2, …, R_N, which follow each other in the file. A file pointer is used to manage operations with this file.

a) Draw a diagram showing how the records are laid out on the file. Include the file pointer's position before any records are read from the file.

(4 marks)

b) Describe how the END-OF-FILE (datafile) condition is detected.

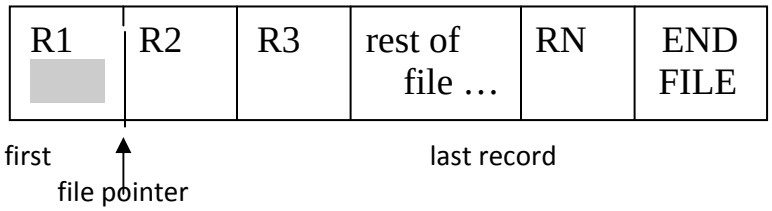
(3 marks)

c) Write a program loop which opens the file, counts how many records are on it, then closes it. State which language you use.

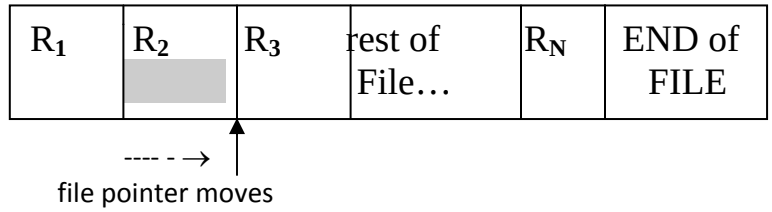
(5 marks)

Answer Pointers

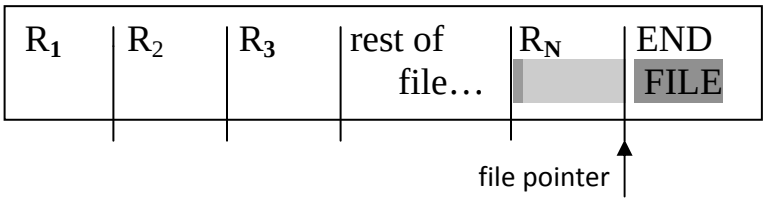
(a)



This shows the file and associated pointer after executing the 'reset' instruction.
Note that the file pointer indicates the start of the NEXT record on the file that is available.
After executing one READ (datafile) instruction the file pointer moves down the file by one record, as shown below:



(b) End-of-file becomes TRUE when the LAST record has been read. On most systems the actual end-of-file marker is not read over. An error condition results if another READ operation is now attempted.



(c)

```
OPEN datafile for READING
recordct ← 0
WHILE NOT END-OF-FILE  DO
      READ (datafile, onerec)
      ADD 1 TO recordct
ENDWHILE
CLOSE datafile
PRINT "the number of records is " recordct
END
```

Examiner's Guidance Notes
Nearly all students gave a linked list diagram. While some systems implement files in this way, they are usually different in how they are used and coded. Only a few could provide a simple loop, terminated by end-of-file (EOF) to read the usually unknown number of records on the file.

Question B6

The following data is stored for cricketers:

Family Name	20 characters
Given names	20 characters
Nationality	10 characters
Type of player	batsman or bowler or keeper
batting average	real number
bowling average	real number
played in more than 10 matches	yes/no

a) Devise a record structure named **player** to hold this data.

(3 marks)

b) Write a program which reads the serial file **cricketers** and which prints in a table those who have not played in any test matches, have a bowling average less than 20 and a batting average greater than 40. A count of these players is also made and printed at the end.

(9 marks)

Answer Pointers

(a) player = RECORD
Last_Name, Other_Names : longstring;
Nationality : shortstring;
Type : (batsman, bowler, keeper);
batting_ave, bowling_ave : REAL;
test_matches : INTEGER
END;

(b)

```
(* table caption *)
PRINT "    name    bowling average batting average"
ctrec = 0
OPEN cricketers for READING
WHILE NOT EOF (cricketers DO
        READ(cricketers, onerec)
        IF onerec.test_matches = 0 THEN
            IF batting_ave > 40  AND bowling_ave <  20  THEN
                PRINT onerec
                ctrec = ctrec + 1
ENDWHILE
PRINT " number of players selected=" ctrec
CLOSE cricketers
```

Examiner's Guidance Notes

A popular question, most answers had only the record declaration. Very few knew about condition controlled loops needed here. A worse mistake was including the ellipsis (…) in code. Answers usually contained [uses crt, clrscr, #include <iostream.h>] which was not required.

Question B7

Values for the hyperbolic cosine function are obtained from the power series

Cosh(x) = 1 + x^2 / fac(2) + x^4 / fac(4) + x^6 / fac(6) + …
where fac(n) = factorial n = 1*2*3*4*…n

a) Write code for fac(n); any method may be used.

(4 marks)

b) Incorporate your function into another function Cosh(x) which calculates Cosh(x) using the power series given earlier. Show how to terminate the calculation when the difference between successive terms is less than 0.00005.

(8 marks)

Answer Pointers

```
FUNTION fac(n:INTEGER):INTEGER;
IF n = 0 THEN fac = 1
        ELSE fac = n*fac(n – 1)
ENDFUNC


sum = 1
n = 0
mult = 1
WHILE not end condition
        n = n + 2
        mult = mult * x * x
        term = mult / fac(n)
        sum = sum + term
ENDWHILE
```

Examiner's Guidance Notes
Usually answered by those who had memorized the code for factorial(n). Some attempted mathematical derivations outside the syllabus. Again students did not know how to set up a condition-controlled loop. Marks were awarded for those who coded specifically the terms given in the question.

Question B8

Bank account details are stored in a linked list and are such that each entry takes thefollowing form:

Name (20 characters)
Account number (integer)
Balance of account (real number)
Negative balances indicate overdrawn accounts.

a) Provide suitable definitions for such a linked list. State which language you are using.

(3 marks)

b) Write a program which will print a table detailing all accounts from the linked list that are overdrawn. The name, account number and amount in debt should be printed under a suitable caption. Afterwards, a count of how many accounts were overdrawn must be printed.

Make appropriate variable declarations for part (b). Do NOT repeat the record declaration made in part (a) but show where it would be placed.

(9 marks)

Answer Pointers

(a)

```
TYPE     ptr = ^node
         node = RECORD
           name :ARRAY[1..20] OF CHAR;
           acount_number : INTEGER;
           balance : REAL
           next as ptr
         ENDREC

VARIABLES head, member as ptr
```

(b)

```
PROCEDURE setup code here

PROCEDURE printlist(ref as ptr)
      PRINT ref^.data, spaces
      IF (ref^.next NOT EQUAL TO NIL) AND (balance LESS THAN 0.0) THEN
          PRINT name
          PRINT account_number
          PRINT balance
          printlist(ref^.next)
ENDPROC

BEGIN  (* top level *)
PRINT 'BANK ACCOUNT PRINTING PROGRAM');
Head ← NIL
setup(head)
PRINT “OVERDRAWN  ACCOUNTS”
PRINT “    NAME                          ACCOUNT NUMBER       DEBT”
printlist(head)
ENDPROG
```

Examiner’s Guidance Notes

Again many answers had only the record declaration, shorn of the pointer fields. Others made a file declaration here. There is clearly confusion among candidates about linked lists and files. Many quoted code for interactive input, although this was not asked for.

Question B9

One of the common operations required of a computer program is to sort items into ascending (or descending) according to the value of a key. Describe (either in words, or by pseudo code or by actual program code) one algorithm for sorting.

(12 marks)

Answer Pointers

The bubblesort was the most popular sort.

mention array (e.g. v)
mention length of array (e.g. len)
concept of adjacent elements (e.g. v[i], v[i+1])
sweep through array swapping adjacent elements if out of order

```
for(i=0; i<len,i++)
        if(v[i]>v[i+1]){…code to swap v[i], v[i+1]…}
```

repeat sweep until no more elements are swapped

Examiner's Guidance Notes
The question clearly requests an algorithm, so an example (trace) of how bubblesort works was only awarded 4 marks (if correct)

For an answer with an algorithm 2 marks could be awarded for each of
- introduction of array
- loop 1
- nested loop 2
- conditional/if
- adjacent elements v[i], v[i+1]
- 3 step swap

The 3 step swap code was often written badly - common mistake were that the assignment statements were written backwards (y:=x instead of x:=y) or the 3 steps were in the wrong order or there was no bracketing around the 3 statements (BEGIN...END or {...})

Some candidates again wasted time in writing code to read values into the array. This is not part of the algorithm of sorting.

The question was misunderstood by some candidates so that (wrong) answers offered: binary chop, binary search, the definition of an algorithm, reversing the order of an array (to get from ascending to descending order)

Question B10

Compare the following pairs of terms. [*You are advised that three well chosen sentences per pair will be sufficient - one sentence describing the first term, one sentence describing the second term and a final sentence highlighting the difference between the terms.*]

a) Compile-time error and run-time error

b) Sequential access file and direct access file

c) High-level language and low-level language

d) Sequential and parallel programming

(12 marks)

Answer Pointers

a)
Compile-time error: syntax error, grammatical error in program
Run-time error: computational error, divide by zero, infinite loop
Difference: compile time errors always come before run-time errors

b)
Sequential access file: pointer in file, only forward progress allowed, get to desired point via every record on the way. If gone too far then only option is to rewind and start again from beginning.
Direct access file: can go directly to any record in file irrespective of where the last visit was made
Difference: sequential access simpler but slower

c)
High-level language: programmer insulated from properties of machine (store size, word size, actual memory addresses). Can define named variables and data structures and use sophisticated control structures
Low-level language: m/c language or assembly language. Have to deal with actual m/c instructions and registers and store addresses.
Difference: HLL suits humans, LLL suits computer. Can get HLL auto translated to LLL

d)
Sequential programming: One thing at a time. Program states exactly what order process are to be executed in
Parallel programming: multiple tasks executing at the same time
Difference: sequential more common, easier to learn. Obvious communication between tasks – new task inherits everything left by previous task. Communication between parallel tasks is tricky – even working out whether a group of parallel tasks have all finished can be awkward.

Examiner's Guidance Notes
A popular question. Often the quality of the 4 parts was very uneven.

a) Some candidates discussed a compiler and an interpreter rather than compile-time error and run-time error. Many candidates didn't write the difference between the two error types.

b) The terms were mostly described well. Some candidates wrote about the memory access or directory structures rather than file access types. Some gave an example about reading data from CD and Tape. They were given some marks.

c) Many candidates gave a satisfactory answer for the question. A few candidates couldn't describe the differences between HLL and LLL. Some have given too brief descriptions about the terms.

d) Unfortunately, many candidates had a wrong understanding of "Parallel programming". The parallel programming is not a parallel development of an application. Likewise, the sequential programming is not a sequential development of an application.

Question B11

One particular software development method is named the waterfall method.

a) Write out the names of all the phases in the method

(3 marks)

b) Choose THREE phases and write a short description of each of the three phases you have chosen

(9 marks)

Answer Pointers

According to some the waterfall method has 5 Phases: Requirements, Specification, Design, Implementation, Testing
According to others the waterfall method has 7 Phases: Feasibility, Analysis/Requirement, Design, Coding, Testing, Implementation, Maintenance/Review

Then THREE from
Requirements: gathering information about the behaviour of the system
Specification: writing down the required behaviour and getting agreement of customer
Design: choosing effective data structures and algorithms
Implementation: writing final code solution
Testing: choosing a deliberate testing strategy e.g. by module, white-box, black-box, unit testing, system testing, etc

Examiner's Guidance Notes
Some candidates wasted time by writing out descriptions of ALL the phases in the method. Only the first 3 were marked.

As ever in this type of question where a definition is required, it is always a sign of a poor answer not to be able to find alternative or synonym words, so "**Design** is the **designing** of ..." is a bad way to start an answer

Note that the question does not ask for an opinion to be expressed about the most important phase, so "Design is the most important phase of the method..." is a bad way to start an answer.

It is not a definition of a stage to say that it comes after ... or comes before ..., it is what goes on within the stage that is important.

Question B12

Describe the general concept of black box testing and illustrate your answer using the subroutine reverse() whose task is to reverse the elements of array **v** as shown below.

index	0	1	2	3	4	5
v (initial values)	2	4	6	8	10	12
v (after reversal)	12	10	8	6	4	2

(12 marks)

Answer Pointers

Black box testing a process of testing a subroutine (procedure or function) without full knowledge of the code, so that the testing checks that results for particular input match the specification.

Test might proceed as follows;

- prepare test data in testv
- copy testv to v
- run the subroutine
- now check the following

```
for(i=0; i<length(v); i++)
        if(testv[i] != v[length(v)-1-i]) report('error')
```

Examiner's Guidance Notes
The question was unpopular and poorly done. Often the last question attempted as many answers were clearly left unfinished. Many have described the general concept of Black Box testing well. A few candidates attempted to write the "reverse" function which was not required. Some wrote a comparison with White Box testing.