1.1 Programming part 3 Mark schemes.

Q1.

(a) Do not use any global variables// use only local variables and/or parameters;
A Can be compiled independently // can be placed in a library

1

(b) Local and global variables with the same name can be misidentified//difficult to test individual procedures/ functions//
Not clear that as a side-effect of executing procedure/function a global variable could change its value;

1

[2]

Q2.

(a) It calls itself / is defined in terms of itself / contains within its body a reference to itself;

Ensure 'it' refers to procedure, if meaning program or object no mark

1

(b) The current state of the machine is saved/preserved;
So can return <u>correctly</u> (to previous invocation/call of **Process)**;
OR
Return address / procedure parameter / status register / other register values / local variables must be saved/preserved;
So can return <u>correctly</u> to "*correctly' can be implied* (previous invocation of Process);

2

(c) Printed Output:
1; 3; 5, Bird; Bremner; 4, Fortune, Jones; 2, Smith;

Mark from left and stop marking when error encountered
Ignore punctuation.

6

(d) (in-order) traversal of a tree; **A** printing of tree (elements in order)
I wrong order

1

[10]

Q3.

(a) Integer;(1) **R** long integer
Whole numbers only // cannot have fractions // discrete number;(1)
R not a decimal number

2

(b) Text / string / alphanumeric;(1)
R numeric
Need a string as number would lose leading zero // not for calculations // because telephone numbers can contain <u>non-digit</u> characters;(1)

2

(c) Boolean // yes-no // logical // subrange with 2 possible values only; (1)
R tick box
R string[1]
R character
Only two possible values;
It's a yes/no answer; (1)

2

(d) Real // single // floating point // float // fixed point; (1)
R double
R decimal
Average may be a fractional number // need decimal point / places; (1)
R decimal/decimal number on its own (n.e.)
A answer may not be an integer/integral/whole number

Reason marks independent of data types

2

[8]

Q4.

(a) (i) Var S: String // Var Count: Integer // Var Size: Integer;

1

(ii) If Size > 0 // If Size >0 Then // If Size > 0 Then EndIf ;

1

(iii) S := "fred" // Size := Length(S);

1

(iv) For Count := 1 to Size Do // For Count := 1 to Size Do EndFor;

1

(b) (i)

Subroutine	Procedure	Function
Length		✓
ToUpper	✓	

2

(ii) Function returns a value // function has a (data) type //
Function appears in an expression //
Function appears on the RHS of an assignment statement;
Procedure does not have to return a value //
Procedure forms a statement on its own;

2

[8]

Q5.

(a) (i) Const MaxChars=5;

1

(ii) Message : Array[1..MaxChars] Of Char//
LastChar : Integer//
Position : Integer//

Found : Boolean;
I var

1

(iii) Position := Position + 1//
Message[Position] := c//
Position := 0//
Found := FALSE//
Found := TRUE//
Find := Position//
Find := 0;

1

(iv) If LastChar < MaxChars (Then ….)//
If Message[Position] = c (Then ….)//
If Found (Then …..);

1

(v) While (Position < LastChar) And Not Found (Do …);

1

(b) (i) Found;
R var
I type

1

(ii) Message// LastChar// Position;
R var
I type

1

(iii) c;
I type

1

[8]

Q6.

(a) A procedure/routine which calls itself//is defined in terms of itself;
R re-entrant
A function instead of procedure
R program iteration Talked Out (no mark)

1

(b) (i)

E	L	H	M	List[M]	Printed Output
6502	1	11 ;	6	5789 ;	
6502	7	11 ;	9	8407 ;	
6502	7	8 ;	7	6502 ;	
					True;

Accept True in row 3

Marks in each row for all three/two parts correct
Accept empty cell to mean: same as in previous row.
Stop marking when logic goes wrong

7

(ii) Binary search;;
Search;
R any other type of search

2

[10]

Q7.

(a) It calls itself / is defined in terms of itself / is re-entrant / contains within its body a reference to itself;

Ensure 'it' refers to procedure, if meaning program or object no mark

1

(b) The current state of the machine must be saved/preserved so can return correctly to previous invocation of B;
OR
Return address / procedure parameter / status register / other register values / local variables must be saved/preserved so can return correctly to previous invocation of B**)**;

1

(c)

Call Number	Parameter
1	53
2	26
3	13
4	6 ;
5	3 ;
6	1 ;

Printed Output: 1 1 0 1 0 1;;;

1 mark for each correct pair of bits
Mark from left and stop marking when error encountered ignore punctuation. if more than 6 bits give a max of 2 marks

6

(d) Conversion (of a denary number) into binary;

1

[9]

Q8.

(a) (i) Var S1: String / Var S2: String / Var Ptr : Integer / Var L :

String; 1

(ii) IF S1 = S2;

1

(iii) For Ptr := 1 To 3 Do;

1

(b)

subroutine	procedure	function
copy		Y;
concat		Y;
print	Y;	

3

(c)

S1	Ptr	L	S2
"PAT"			""
	1	"P"	"P"
	2	"A"	"AP"
	3	"T"	"TAP"
Printed Output:			**False**

8

If S2 at end contains "PAT" then f.t. for True;
(If S2 does not contain "TAP" check that printer output is correct
Depending on what is in S1 and S2 in the candidate's answer)
1 mark for each correct entry, 1 mark for S1 correctly left as "PAT" or empty

[14]

Q9.

(a) *Any two at two each; If entrance method doesn't match exit method mark one wrong and the other correct*
R Voice **R** Written to ticket
Computer system/Printer prints number on ticket at entrance;
Driver types number into system using a keypad at exit barrier;

Computer system encodes number on a magnetic stripe on ticket at entrance;
R Magnetic card
Ticket number read by a magnetic stripe reader at exit//inserted into a magnetic stripe reader at exit; **A** magnetic strip/stripe scanner

Computer system/Printer prints number printed on ticket at entrance;
Number read by an optical character reader/OCR at exit//ticket inserted into an optical character reader at exit;

Computer system/Printer prints number in barcode form on ticket at entrance;
Number read by barcode reader at exit//ticket inserted into barcode reader at exit;

Computer system/Printer at entrance punches holes on ticket which are a coded form of number//Kimbal tag produced at entrance which encodes number;
Number read by sensor (mechanical or optical) at exit//ticket inserted into sensor at exit//Number read by Kimball tag reader at exit;

Computer system/printer prints number using magnetic ink;
At exit MICR reader reads number;

Computer system/printer prints marks (encoding number) on ticket;
At exit, OMR device is used;

4

(b) **R** any other data types. Mark is for field name + correct data type.

NB synonyms for RandomNumber must include Number, e.g. IDNo, TicketNo, Number. **A** RandomInteger, **R** e.g. Vehicle ID **A** VehicleIDNo

A DateTicketWasIssued

```
Record

  RandomNo : Integer;
```

R anything else **1**

```
CurrentDate :
    String/Date/DateTime/TDateTime/TDate;
```

1

```
  ArrivalTime :
 String/Integer/Time/DateTime/TDateTime/TTime;
```

1

```
  LengthOfTime/LengthOfStay/TimeStayed : Integer;
```

R anything else **1**

```
  Cost/AmountToPay : Integer/BCD;
```

1

```
End;
```

A Alphanumeric for String
R Text **R** LeavingTime **R** Binary,Byte,LongInteger
R Date for FieldName
R Date/Time but don't penalise twice

[9]

Q10.

(a) Head (Tail (Days)) = Mon
R [Mon], MON (1)

Tail([Head(Days)]) = [ ] (1)

Empty(Tail(Tail(Tail(Days))))=False (1)

3

(b) Elements in a list can only be accessed sequentially;
elements in an array can be accessed directly;

using the subscript;

Any 2 points

2

[5]

Q11.

(a) (i) Const Max = 200;

1

(ii) EndOfList := False / Ptr := 1 / EndOfList := True / Ptr := Ptr +1;
A without :

1

(iii) If Ptr > Max Then …../ If EndOfList (Then) …..;

1

(iv) While WantedName < > Member[Ptr].Name And Not EndOfList Do …;
A While …. End While;

1

(b) (i) Tmember;
A (Type) Tmember = Record;

1

(ii) WantedName;
A WantedName: String;
R *whole line*

1

(iii) EndOfList; **A** (Var) EndOfList: Boolean;

1

(c) When the programmer wants to change the value it only needs changing in the declaration;
Can't be changed accidentally/by the program; easier to understand/ debug program;
A less error prone;
R easier to read;

1

(d) (i) Because the age would need to be manually updated when it's someone's birthday;
A an answer which implies age changes value; **I** lack of accuracy

1

(ii) Store the date of birth and calculate the age from that and today's date;
A date of birth *on its own*

1

(e) TRUE and FALSE / 1 and 0 / 0 and –1 / on and off / Yes and No / high and low;

1

[11]

Q12.

(a) (i) Assembler;

1

(ii) Interpreter / compiler;

Max 1

(b) Problem oriented;
Portable; machine independent;
One-to-many mapping of HLL statement to machine code statement;
Datatypes; structured statements; local variables; parameters; data structures;
A *example of a datastructure;*
Named variables; named constants; English-like keywords/commands;
Quick/easy to understand / write / debug / learn / maintain;
R quick/ easy to use
R modular
R procedures
R takes longer to translate
R closer to English

Max 2

(c) Easier to understand / more transparent; less error prone; easier to maintain / change (if the value changes);

Allow by example
eg if VAT rate changes only need to change value in declaration

Max 1

(d) (i) Accept any imperative HLL such as Pascal/VB/C/C++/PL1/Cobol;

SEE TABLE FOR DIFFERENT LANGUAGE EXAMPLES FOR (ii) & (iii)
Ignore line breaks in statements

1

(ii) *1 mark for correct key words in correct order (shown in bold in table overleaf); 1 mark for correct Boolean expression / loop control expression ;*

2

(iii) *1 mark for correct key words in correct order (shown in bold in table overleaf); 1 mark for correct boolean expression;*

2

If (i) does not match (ii) and (iii) do not give marks for (ii) and (iii)
If candidate names a language you are not familiar with, contact your team leader

Language	Bool expr	Iteration *....are possible statements(ignore)*	Selection *....are possible statements(ignore)*
Pascal Delphi Kylix	a>b a=b a< >b a<b a>=b a<=b	FOR <variable>:= <value1> TO <value2> DO ..... REPEAT ..... UNTIL <bool expr> WHILE <bool expr> DO	IF <bool expr> THEN ..... *else part optional* CASE <variable> OF <value1>..... <value2>..... ENDCASE *else part optional.* *No of values can vary*
Visual Basic VSScript	a>b a=b a< >b a<b a>=b a<=b	FOR <variable> = <value 1> TO <value2> ..... NEXT DO WHILE/UNTIL <bool expr> .....LOOP DO	IF <bool expr> THEN ..... END IF *Else part optional* SELECT CASE <variable> CASE <value1> .....

		LOOP UNTIL/WHILE <bool expr> WHILE <bool expr> ..... WEND	CASE <value2> ..... End Selct *Else part optional* *No of CASE values can vary*
C/C++ Java Javascript	a>b a==b a!=b a<b a>=b a<=b	FOR (<initialisation>; <condition>; <increment>)..... WHILE (bool expr) DO	IF (bool expr) { } *Else part optional* SWITCH ( ) {CASE <value>: BREAK} *No of CASE values can vary*
COBOL	a>b a=b a< >b a<b a>=b a<=b	PERFORM .....<number> TIMES PERFORM VARYING <variable> FROM <value>BY <value> UNTIL <bool expr using variable>	IF <bool expr> PERFORM ..... *Else part optional*
Fortran	a.LT.b a.GE.b a.LE.b a.GT.b a.NE.b a.EQ.b	DO <number> < variable>= <init value> <final value> *step value optional*	IF <bool expr> ..... IF (<arithmetic expr>) label1, label2, label3
Basic	a>b a=b a< >b a<b a>=b a<=b	FOR <variable> = <start value> TO <stop value> .... NEXT <variable> *step value optional* REPEAT .... UNTIL <bool expr>	IF <bool expr> THEN ..... GOTO label1, label2, label3 DEPENDING ON <variable>

2

[10]

Q13.

Reason mark not dependent on correct data type

(a) Integer/byte; **R** long integer
Whole numbers only/ can **NOT** have fractions/ discrete number;
R not a decimal number

2

(b) Text / string / alphanumeric; **R** numeric
Need a string as number would lose leading zero / not for calculations / because telephone numbers can contain non-digit characters;

2

(c) Boolean / yes-no / logical / subrange with 2 possible values only;
R tick box
R string **[1]**
R character

Only two possible values; it's a yes/no answer;

2

(d) Real / single / floating point / fixed point
R double
R decimal.

Average may be a fractional number / need decimal point/places
R answer may not be an integer/integral/whole number
A answer may not be an integer/integral/whole number

If number already implied from data type given;

2

[8]

Q14.

(a) To give a set of statements a name;
To avoid repeating code; procedures can be called up many times; so a program may be developed by more than one programmer;
To enable top-down development;
To enable stub-testing; so libraries of code written by other programmers may be used;
For creating libraries of code;
Allows for modular programming; allows for modular testing;
To make program easier to debug / maintain / understand;

Max 2

(b) To pass data/value within programs / to pass data/value in and out of procedures/functions;

1

[3]

Q15.

(a)
```
CONST max=5;
```
1

(b)
```
VAR Tptr: INTEGER/
VAR store: ARRAY[1..max] OF CHAR/
VAR ptr: INTEGER;
```
1

(c)
```
ptr;
```
1

(d)
```
Tptr/store;
```
1

(e)
```
a;
```
1

(f)
```
Tptr := Tptr +1/
store[Tptr] := a /2
take := store[1]/
Tptr := Tptr -1/
store[ptr]:= store[ptr+1];
```
1

(g)
```
IF Tptr<max THEN…./
IF Tptr>0 THEN…..;
```
1

(h)
```
FOR ptr:=1 TO Tptr DO …..;
not case sensitive
```
1

[8]

Q16.

(a) Tail(Ports) - [Barcelona, Athens, Alexandria, Tunis, Lisbon]

square brackets needed

1

Head(Tail(Tail(Ports))) – Athens (2)

[Athens] (1)

2

Empty(T(T(T(T(T(T(Ports)))))))) – True (2)

True [ ] (1)

[True] (0)

2

(b) **Recursively defined**
A definition which is defined in terms of itself/contains within its body a reference to itself/calls itself ;
A re-entrant; (In specimen papers 2001/2, but refers specifically to a procedure)

1

(c) **Stack necessary**
The state of the machine/contents of appropriate registers/ return address // saved each time the procedure is called (1) and retrieved in reverse order from the stack as control is progressively returned (1)
OR
Different value of parameters /local variables (1) must be available each time procedure is called (1)
OR
P must be re-entrant (In specimen papers 2001/2)(2)

2

(d) Lisbon first (1)
Southampton last(1)
All 6 in order (1)
No punctuation (1)

i.e. Lisbon Tunis Alexandria Athens Barcelona Southampton;

4

(e)

1	Southampton	5	
2	Barcelona	6	Head Pointer
3	Athens	2	4
4	Alexandria	3	
5	Tunis	0 - terminator	
6	Lisbon	1	

2

[14]

Q17.

(a) Easier maintenance/upgradeable; can get an overview of system;
Quicker to write/Easier development; can break problem down into sub tasks/ can re–use modules/ distribute among team;
R easier to read so can get a team to write program;
More systematic testing; can test a module at a time;
Fewer mistakes made; clear organisation of code;
Quicker/easier to debug; easier to see where errors are;
Fewer lines of code using subroutines; code not duplicated;

(1 mark for reason, 1 mark for explanation
OR 2 marks for good explanation per point)

Max 4

(b) Structured statements such as iteration / selection;
Use of Procedures / functions/subroutines/modules/libraries;
User defined data types; built–in data types; Data structures;
Can choose sensible names for identifiers;
English–like keywords / constructs; Indentation; Comments;
Use of local variables; parameters; named constants;
R closer to natural language/ almost written in English
R machine independent / problem oriented / top–down

Max 3

(c) Compiler translates whole source code into object/machine/runnable code / exe file;
Compiled program runs faster than interpreted program;
Interpreter translates line by line as it executes/is running;
Interpreter must be in memory to execute program; compiler only needed during translation stage, not during execution of program.;
interpreter runs one line at a time;

Max 2

[9]

Q18.

(a) Integer/Byte (1)
Whole numbers only / can not have fractions / discrete number (1)

2

(b) Real / single / floating point / fixed point; **R** double (1)
Average may be a fractional number / need decimals (1)

2

(c) Text / string (1)
Need string as number would lose leading zero/ not for calculations / because telephone numbers can contain non-digit characters (1)

2

[6]

Q19.

(a) (i) VAR/CONST/TYPE/DIM/FUNCTION/PROCEDURE/LABEL
Or similar, name and type;; keyword and name;;

2

(ii) Eg x:=5 / y ← y – 1

1

(iii) Example of IF / CASE / SWITCH statement

(1 mark for keyword, 1 mark for selection criteria)

2

(b) (i)

	S		S	
	D	[5]	M	[5]
	K	[4]	T	[4]
	C	[3]	C	[3]
	T	[2]	K	[2]
	M	[1]	D	[1]

1 mark for each correct entry

10

(ii) Algorithm: reverse content of array
R re-arrange

1

[16]

Q20.

(a) (i) Value passed into, or out of, procedure from / to calling program

1

(ii) Variable defined outside any procedure and so in scope for whole program

1

(b) Saves memory since parameter takes no space when not in use
Enables portability of procedures / blackbox programming / incorporation of procedures into library
Enables re-using of variable / identifier names for formal parameters without interfering with other procedures
Passing by value prevents accidental alteration of a value having unexpected side-effects in other procedures
1 mark for a relevant point

1

[3]

Q21.

(a) Causes process to repeat indefinitely

NOT repeats until maintain is TRUE

1

(b) Maintain has two values, TRUE / FALSE (1), => must be Boolean (1) n is used as an array subscript (1) => must be integer (1) or n is used as a loop control and can never be non-integer within the algorithm (1) => integer (1)

NOT numeric - too vague

4

(c) See table for model solution 1 mark each indicated section completed correctly, including follow–through (7x1); additional 1 mark for correctly modifying n downwards in penultimate section If candidates go completely wrong but clearly deserve some credit marks can be awarded on the following criteria, up to a maximum of 2 marks for correct sequence of loop repetitions, including the change from 6 to 5 then 6 - i.e. the column for n, including correct exit 2 marks for correct completion of the sequence of stations, ie the org, dest, start, finish columns

2 marks for correct completion of totalkm column, i.e. correct lookups and totalling 2 marks for correctly executing inner if branches, i.e. setting maintain and resetting totalkm in correct places. Total 8 marks for all-correct trace follow-through marks should be awarded where appropriate

8

(d) 2 marks for diagram, or explanation, showing that the journeys indicated above cover all routes in both directions *marks can be awarded for any reasoned answer (indicating achievement or not) providing it is consistent with the candidate's trace table* Note: the sequence is MK -> SW -> CW -> SW -> TW -> HK -> MK -> QB -> SW etc., which does cover all lines in both directions. Strictly speaking, whether the objective is achieved depends whether journeys to/from MK depot are passenger-carrying / revenue-earning or not. Either interpretation is acceptable - the marks are awarded for the explanation.

n	org	dest	last	start	finish	totalkm	maintain	Remarks
	0	3	1					
							FALSE	
				MK				
					SW			
						15		
	3							
0								

Given

1								
		4						if ignored
				SW				if ignored
					CW			
						+27 = 42		
	4							N<6 so rpt

1 mark

2								
		3						if ignored
				CW				if ignored
					SW			
						+27 = 69		
	3							N<6 so rpt

1 mark

3								
		1						if ignored
				SW				if ignored
					TW			
						+37=106		
	1							N<6 so rpt

1 mark

4								
		5						if ignored
				TW				if ignored
					HK			

						+34=140		
	5							N<6 so rpt
5								

1 mark

		2						if ignored
		0						>140 so if executed
			5					
							True	
				HK				
					MK			
						+12=152		
	0							N<6 so rpt

1 mark

6								TRUE so if executed
5								
						0		
							False	
		2						
								if ignored
				MK				
					QB			
						+28 = 28		
	2							N<6 so rpt

1 mark

6								

		3						if ignored
				QB				if ignored
					SW			
						+43 = 71		
	3							
								N = 6 so stop repeat loop
								end while

1 mark

2

[15]

Q22.

(i) 10110000

1

(ii) 00110010

1

(iii) 00000001

1

(iv) 10001011

1

[4]

Q23.

(a) 00001000

1

(b) Or / XOr

1

[2]

Q24.

Statements must be sufficient to demonstrate general principle e.g.

(a) X: =Y+Z;

1

(b) IF X>I0 THEN PRINT “GOOD”../CASE OF / SWITCH;

1

(b) WHILE X<10 DO.X=X+1../REPEAT...UNTIL X=I0/FOr X=I TO 10 DO;

1

1 mark for each example
Max 3

[3]

Q25.

(a) Variable with only two possible values, e.g. true/false, on/off

1

(b) (i) FALSE

(ii) TRUE

LY = F AND (NOT F OR F) = F AND (T OR F) = F AND T = F

LY = T AND (NOT T OR T) = T AND (F OR T) = T AND T = T

1 mark each

2

[3]

Q26.

(a) Information passed to / from a function or procedure to define the values it is to use - e.g. in calling OPENSCREEN, the values "Admin Computer" and 10 [actual or real parameters] are to be used as the values of COMPUTERNAME and CHANNEL [formal parameters or placeholders]

3

(b) Enables same function to be used in a number of contexts, enables "black-box" programming, or enables different programmers to work on various modules

Accept: saves memory by not using global variables, or prevents inadvertent modification of variables in a procedure

2

(c) As a series of contiguous / consecutive memory locations

1

(d) Extract a character from the MSG array, at the position indicated by COUNT
Call the SENDCHARACTER function, passing it CH and other data Increment COUNT and COL
Repeat the process as long as CH does not have the value 13 [4 important points are: repetition, what condition determines whether to repeat (examine current value of CH variable), if condition is true what happens (execute block to endwhile), effect of instructions in loop (next character in sequence sent to other computer and screen coordinates adjusted)]

4

(e) Prints one of the specified messages, depending on the value of ERR

2

(f) Would be too long for the MSG array and so might overwrite other data or code
Accept: data truncated, interpreter produces runtime error (array bounds

exceeded), compiler error causes program build to abort

1

[13]

Q27.

See trace table below. Sections corresponding to marks are shaded.

1 mark for newstring, message and procedure call correct. 1 mark for x and piece correct. 1 mark for outstring correct. 1 mark for changing x and a. 1 mark for tracing the second call to docharacter. 1 for section correct. 1 mark for third call correct. 1 mark for endprocs all correctly traced. 1 mark for outputs correct

		↓						↓	Trace table				↓						
	New string	message	a	Out string	x	piece	x>0?	a	Out string	x	piece	x>0?	a	Out string	x	piece	x>0?	output	marks
Input message		CAT																	
New string: = “”	‘’’																		
Output message																		CAT	
Docharacter (message, new string)			CAT	‘’’															
x :=1en(a)					3														
Piece := Right$(a,1)						T													
Outstring:=outstring+piece				T															
x := x-1					2														
If x>0 then							true												
a= Left$(a,x)			CA																
Docharacter (message, new string)								CA	T										
x :=1en(a)										2									
Piece :=Right$(a, 1)											A								
Outstring: = outstring+piece									TA										
x :=x-1										1									
If x>0 then												true							
a= Left$ (a, x)								C											
Docharacter (message, new string)													C	TA					
x :=1en(a)															1				
Piece :=Right$(a, l)																C			
Outstring: = outstring+piece														TAC					
x :=x-1															0				
If x>0 then																	false		
Endif																			
endproc								C	TAC										
Endif																			
endproc			C	TAC															
Endif																			
endproc	TAC	C																	
Output new string																		TAC	

Note that there is no need to trace a and outstring separately in each cell as there are parameters passed by reference. If they are included in the trace then recursive calls must be shown. X and piece must be shown as additional columns for each cell.

[9]

Q28.

(a) (i) 00000010
AND

1 for mask. 1 for AND

2

(ii) 10000000
OR

1 for mask. 1 for OR

2

(b)

```
Repeat
     Test active
     If active = true then
       Test external motion sensor
       If external motion sensor = true then
            security light: = on
       else
            security light: = off
       endif
       Test internal motion sensor
       Test window contact
       Test door contact
       If ((internal motion sensor = true)or (window contact =
false)or
       (door contact = false)) then
            Alarm: =on
       else
       Alarm off
       endif
     endif
until set = false
```

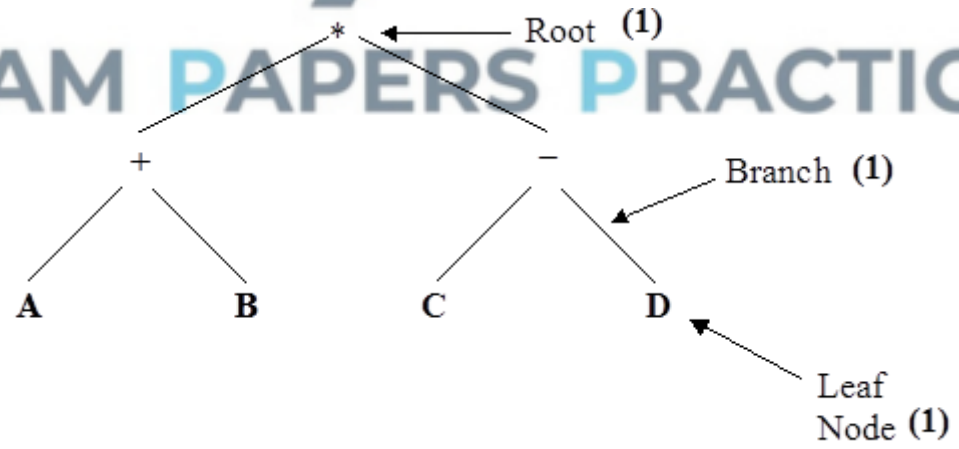
1 mark for suitable loop including termination.
1 mark for testing set.
1 mark for If correctly used with endif.
1 mark for testing external sensor and handling light.
1 mark for testing all three sensors (and alarm on.)
2 marks for a single if construct, just 1 if there are 3 separate ifs

7

[11]

Q29.

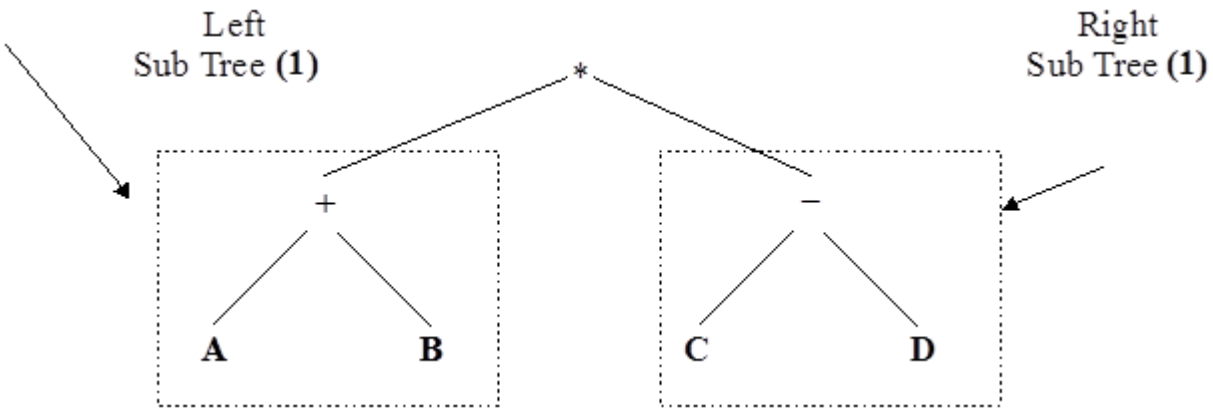
(a)



Labelling must clearly indicate term

3

(b)



Must clearly indicate subsets

2

(c) A procedure which is defined in terms of/ calls itself /re-entrant

1

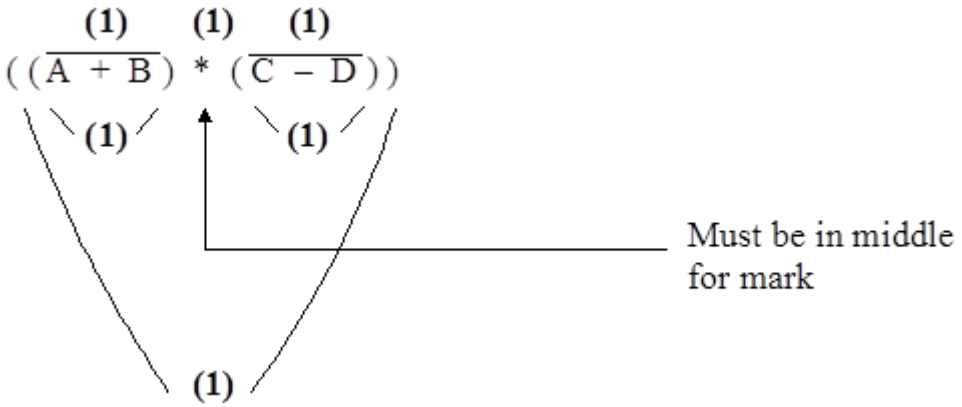
(d) State of machine/return address/parameter (1) needs to be stored/held (1) to enable a previous execution of T to be resumed (1)

Or
So that each call to T(1) can pass(1) a new value of the parameter(1)

3

(e)

Call Number	Parameter	
1	**tree**('*', **tree**('+', **tree**('A', empty,empty), **tree**('B',empty,empty)), **tree**('-', **tree**('C', empty,empty), **tree**('D',empty,empty)),)	
2	**tree**('+', **tree**('A',empty,empty), **tree**('B',empty,empty))	
3	**tree**('A',empty,empty),	
4	**tree**('B',empty,empty),	(1)
5	**tree**('-', **tree**('C',empty,empty),**tree**('D',empty,empty))	(1)
6	**tree**('C',empty,empty),	(1)
7	**tree**('D',empty,empty),	(1)



10

(f) In-order traversal

1

[20]

Examiner reports

Q1.

This question showed that many candidates have not had sufficient exposure to programming techniques.

Parts (a) and (b) often showed up the candidates' inability to express themselves clearly.

(a) Many candidates described recursion. Those that didn't often restated the question. Only the very best candidates were able give a satisfactory answer.

(b) The responses to this part were also very disappointing. Answers were often irrelevant and/or badly expressed.

Q2.

Part (a) and (b) have been asked many times before. However, many candidates were unable to explain how a stack is used in the execution of a recursive procedure. Correct responses included that the return address (held in the program counter) and other register values are saved so control can return correctly to the previous invocation of the procedure. Many candidates gained full marks for stating, correctly, the printed output after dry-running the procedure. However, many others could not even get the first few printed items in the correct order.

Many candidates correctly spotted that the procedure described an in-order traversal of a tree.

Q3.

This question enabled candidates to score well but many candidates were unable to give satisfactory data types.

(a) It was important that candidates recognised that the number of students in a school will be a whole number.

(b) It was pleasing to see the number of candidates who now appreciate that an issue with the storing of telephone numbers is the possible loss of a leading 0.

(c) Most candidates were able to answer this part correctly.

(d) Candidates found this part the most difficult. Even when they realised that the number to be stored would involve fractions and gave a suitable data type they often failed to obtain the second mark. Candidates often gave very poor descriptions.

Q4.

(a) The variable declaration was normally identified. Many candidates found it difficult to identify selection, assignment and iteration statements correctly. This suggests that they have not had sufficient exposure to programming techniques.

(b) It was very disappointing to see how few candidates understand the difference between a function and a procedure. There were many confused and completely incorrect answers. Although there were many correct answers to part (i), few candidates obtained full credit for part (ii). Although many candidates were able to state that a function returned a value, few were able to give the main features of a

procedure.

Q5.

(a) Some candidates failed to obtain marks when they failed to copy complete statements. It was surprising how many candidates were unable to identify an assignment statement and also how many candidates confused selection and iteration.

(b) This part was not answered as well as (a). Few candidates seem to be clear about global variables and even fewer were able to identify a parameter.

Q6.

(a) Most candidates could correctly state that recursively defined means that a procedure is defined in terms of itself or that it calls itself. Some candidates failed to gain marks because they could not express this clearly enough. A common misconception was that the procedure was in a loop.

(b) Many candidates managed to gain full marks for completing the trace table:

E	L	H	M	List[M]	Printed Output
6502	1	11 ;	6	5789 ;	
6502	7	11 ;	9	8407 ;	
6502	7	8 ;	7	6502 ;	
					True;

A common mistake was to have False as printed output in the first 2 rows until it changed to True.

Some candidates who correctly completed the trace table could not see that the process was a binary search and some who did not complete the table correctly did manage to identify the process correctly.

Candidates should be aware of the need to fill in trace tables carefully, showing how values change chronologically. This may mean leaving some cells empty if no values are assigned to variables initially.

Q7.

Those candidates who clearly understood recursion scored high marks in this question, but a worrying number of candidates could not explain that a procedure is recursively defined when it is defined in terms of itself. A stack is needed so that register values such as return address and parameter values can be saved and can be returned to in the correct order. Most candidates managed to complete the trace table correctly but many did not give the correct printed output. Many candidates did not provide the correct number of digits, or in reverse order. A large majority wrongly thought that procedure B described a binary search. Interestingly, some of the candidates who got the printed output wrong still stated the correct purpose of the procedure: converting the denary number provided as parameter into binary.

Q8.

In part (a), a large number of candidates failed to differentiate correctly between a selection statement and iteration.

For (b), very few candidates correctly identified the given subroutines as 2 functions and one procedure. Even though the description of the subroutines in the question stem should have made candidates realise that both *copy* and *concat* were the same type of subroutine, many seemed to hedge their bets and opted for one of each type. Many other candidates got the choice exactly the wrong way round.

In (c), the response to the dry run was much more promising, suggesting that candidates are getting more opportunity to practice this type of skill. A large number of candidates were able to follow the algorithm with only the S2 value causing any problems.

Q9.

Many candidates were able to suggest a suitable method at the exit barrier for submitting the number assigned to the ticket to the computer system. Fewer were careful enough to describe how the ticket was assigned to the ticket at the entrance barrier. The examiners were expecting a printer to be referenced or the action of printing in the case of a barcode, OMR, OCR, MICR and plain text solution for writing a number to the ticket; the action of encoding or writing to the ticket in the case of a magnetic stripe or smart card and the action of punching in the case of Kimball or Kimball-type tags. The better candidates offered such descriptions.

A lack of experience of using a third generation programming language was exposed by part (b). Many candidates gave data types that were lifted straight from Microsoft ACCESS and therefore gained no credit. The question explicitly requested data types that would be available in a third generation programming language. Candidates must understand that their study of this subject must extend beyond products that have been intentionally designed to enable people with no background in Computing, and no desire to be educated in this subject, to achieve practical results in the minimum of time. For very simple practical tasks, this is desirable but as a foundation for more extensive tasks this is a recipe for disaster. Assembling flat-pack furniture is not considered adequate enough to qualify as a cabinetmaker.

Some candidates had difficulty selecting relevant fields and in some cases when they did choose relevant ones lost a mark for poor choice of identifier. For example, several candidates chose *Date* instead of *CurrentDate*. The clue was in the question stem which stated that “the computer system remembers the *Current Date, Arrival Time* and *Randomly generated Number*.“ A principle of software engineering is that identifiers should be meaningful and reflect the real world entities which they represent.

Q10.

Parts (a) (i) and (a) (iii) were answered correctly by most candidates as Mon and False respectively, (not [Mon]). However, in (a) (ii) [Head(Days)} is the list [Sun] and the tail of this list is the empty set [ ]. Had these data been stored in a one dimensional array, instead of in a list, each element could have been accessed directly using the subscript, rather than having to be found through a sequential search or using the Head / Tail functions defined.

Q11.

The evidence suggests that this part of the specification is being lightly treated in some instances since all candidates from some centres appeared to be guessing wildly.

Candidates who had studied, and presumably used, a High Level Language found this question easy – especially if it involved Pascal. Those that had not, struggled.

(a) Many candidates did not follow the instructions 'copy one relevant statement from the above code'. Instead they only gave a keyword, insufficient to earn the mark available. The assignment statement caused most confusion. Many candidates wrongly picked the type declaration, presumably because of the '=' symbol.

(b) Here candidates were asked to copy one relevant part statement. However, many candidates copied whole statements, thereby not making it clear which part was relevant. Very few candidates could identify the user-defined type or the parameter used in the given code. Local variables seemed to be rather better known.

(c) Many candidates correctly stated that using named constants made the program easier to understand or that if its value needed to be changed it would need to be changed in only one place.

Many candidates suggested that the code was easier to read. This was not given credit, as being able to read something is clearly not the same as being able to understand it!

(d) Most candidates spotted that if a person's age is recorded it will have to be changed/updated yearly and therefore it would be better to use date-of-birth. A large number of candidates felt that many people would not like to give their age but then, surprisingly, would be quite happy to give their date of birth!

(e) This part was answered well, with most candidates knowing what values a Boolean variable could take.

Q12.

(a) Candidates often gave two high-level languages, which were inappropriate responses. The question asked for translators and the only correct responses were assembler and compiler/interpreter respectively.

(b) Most candidates gained one mark for the obvious answer that it is easier to write programs in a high level language, without being entirely convincing that they knew what they were talking about. Fewer candidates could list a second characteristic, e.g. that such languages are problem oriented rather than machine oriented, or that they support data structures and structured statements. The response 'can use English words' was not enough to gain credit.

(c) Many candidates gained credit for stating that the use of named constants makes a program easier to maintain or understand.

(d) Those who had, clearly, studied a high level language had little difficulty with this part. Nearly all candidates gained the mark for part (i) since they had heard of a programming language. The responses to (ii) and (iii) however showed that far too many candidates do not get enough exposure to practical programming in a high level language. HTML is not appropriate here.

Q13.

For candidates who had studied a high level language such as Pascal this was an easy question, for those who hadn't it became a lottery with many candidates assuming a data type 'numerical' for parts (a) (b) and (d) with the reason in each case being 'because it's a number'. Candidates should realise that 'numbers' such as telephone numbers need to be

stored as strings as they are not numbers in the mathematical sense of being used for calculations. A significant minority of candidates seem to think that averages should still be expressed as whole numbers 'because you can't have half a car'. Candidates also need to understand that the term 'decimal number' describes a number in base 10, not necessarily a number with a decimal point and fractional part.

Q14.

The few candidates who studied the topic had no difficulty answering this question. Those who had not, often assumed that 'procedure' was a method of programming, 'parameters' were a set of rules or limits beyond which the program could not go. It is important that candidates gain some experience of programming with functions and procedures and appreciate the role of parameters in passing data into / out of these subroutines. A creditworthy response was that the use of procedures avoided repeating code because a procedure could be called up many times from different parts of the program.

Q15.

Parts (a), (b), (d), (f) and (g) were more often answered correctly than the other parts. Very few candidates seemed to understand the term 'parameter'. Surprisingly, 'local variable' was also not very well known. A considerable number also gave an IF… statement as the answer to iteration.

Q16.

(a) This part asked what result would have been returned by the specified function calls. So Tail (Ports) would have returned [Barcelona, Athens, Alexandra, Tunis, Lisbon] with the brackets being part of the correct answer. Head(Tail(Tail(Ports))) would have returned Athens, without brackets and the answer to the last part was True, not [True] or True [ ].

(b) Most candidates could say what recursively defined was, although some answers were barely sufficient and 'It calls itself' was deemed insufficient.

(c) Explanations of why a stack was necessary to execute procedure P recursively frequently missed the point.

(d) Full marks were gained for this part by an answer of:
Lisbon Tunis Alexandria Barcelona Southampton,
or even of:
LisbonTunisAlexandriaBarcelonaSouthampton,
as no punctuation was printed. A list of:

- Lisbon
- Tunis
- Alexandria
- Barcelona
- Southampton

was also accepted.

(e) Candidates who did not score full marks for 10(e) mainly numbered the ports in alphabetical order, rather than using the pointers to point to the next port in the list, or gave inadmissible or absent end-of-list markers.

Q17.

This question was generally done poorly. Most candidates read into the question that a

comparison with machine code or assembly code programming was required. This was clearly not appropriate.

(a) Candidates often quoted the detail of the question again as an answer, but many gained some credit for answers including that it was easier to debug a structured program because it would be easier to find errors in procedures. The misconception that structured code makes it easier for the compiler was rather widespread.

(b) Candidates do not know what features are, with many carrying on answering as in part (a). Often no distinction between the features of an editor used to type a program and the features of the actual programming language itself were made. Most candidates thought that programming in a high level language was almost like writing in English, for which no credit was given. Good answers included use of procedures/functions, English-like keywords, data types, data structures, local variables, parameters, being able to choose sensible names for identifiers and the existence of constructs such as IF..THEN..ELSE, REPEAT..UNTIL etc.

(c) The distinction between a compiler and an interpreter was clearly taught well in some centres and not in others. The fact that a compiler will translate the whole source code into object code, which can then be executed independently of the compiler seemed to evade many. That an interpreter translates one line at a time as it executes without producing object code was also only fully understood by few.

Q18.

Parts (a) and (b) were usually well answered, although a surprising number of candidates confused 'integer' and 'real'. Most candidates appreciated the appropriate data type for whole numbers to be 'integers', and for fractional numbers to be 'reals'. In part (c), however, there did not seem to be a great understanding of the significance of the 0 at the start of the telephone number. This makes a string data type necessary. Candidates from some centres had no idea at all and suggested databases and spreadsheets as data types.

Q19.

Part (a) was always attempted, but many candidates confused a declaration with an assignment. Some candidates tried to answer in general terms rather than with examples asked for. In part (b) some candidates got full marks and demonstrated clearly their ability to follow through an algorithm. Many candidates did not seem to understand how to dry run algorithms; or this was not taught in some centres. Many candidates guessed wrongly that it was a sort of algorithm and simply wrote down the letters in alphabetical order into the array. Some candidates thought that the algorithm reversed the letters and then back again. Most candidates who correctly followed through the algorithm then correctly recognised that the order of the letters in the array was reversed.

Q20.

Parameters are clearly widely misunderstood - many answers reflected the common misunderstanding of the word as "boundaries" rather than context-determining factors. Almost any answer about passing data to / from procedures or functions would do. Most, on the other hand, understood global variables. Answers to (b) were not encouraging, probably because parameters are misunderstood - the best answers were either the use of portable procedures and the ability to include them in libraries for re-use, or the saving of memory by discarding variables not currently in use. Another popular correct answer is the risk of a procedure accidentally altering a global and therefore having unexpected side effects in other procedures.

Q21.

This question divided the candidates into two groups, those who could answer it and those who could not - an encouraging number scored nearly full marks, but an alarming number scored close to zero.

For part (a), although the while (TRUE) construction is standard terminology for an infinite loop few realised it, instead many tried wrongly to relate it to the state of the maintain variable, presumably because it was the only Boolean in sight.

Most candidates understood the data types for (b) - not many gave the correct reason for n being an integer (it is used as an array subscript), but many commented that it was only required to take the values 0 to 6 and so didn't need to be anything else, perfectly valid reasoning. Many students could not even attempt the trace table, but many perfect attempts were seen, conversely some students made marking difficult by making elementary arithmetic errors early on which had to be laboriously followed through to give appropriate credit.

For the last part, which should have demonstrated ability to interpret the table in terms of the original problem, many elaborate flights of logic appeared, which were rewarded provided they were plausible and argued from the student's trace table.

Q22.

Those candidates that were prepared for this question usually got full marks. The XOR operation proved the most difficult.

Q23.

Almost all candidates answered this correctly. Those who did get it wrong seemed to have little knowledge of masking.

Q24.

This question required candidates to give actual examples of programming statements from a high level language with which they were familiar. In too many cases, three easy marks were lost by those candidates who did not read the question and either described the statement or wrote, for example,
IF..... . .THEN. . .ELSE".

Q25.

Most candidates knew what a Boolean variable is but few could manipulate them - many clearly guessed the significance of 1999, 2000 and LY and wrote down the (correct) answers apparently conjured out of thin air. Many candidates clearly could not understand the distinction between 1999 mod 400, which is a numeric expression evaluating to 399, and 1999 mod 400 = 0, which is a statement capable of being evaluated as either true or false: consequently these candidates frequently asserted correctly that a Boolean variable is either 0 or 1, then promptly made the Boolean LY into 399 or other clearly impossible values.

Q26.

This question also asked candidates to relate theory to practical examples to illustrate their understanding of it rather than simply recalling, and in some respects fell down badly. In (a) and (b) the able candidates could describe what the term parameters means and quote examples of where they were mentioned in the pseudocode, but only the very

ablest described their use: a good answer would have been something like “parameters are values passed into subprograms, such as the variable values col and row being passed into the SendCharacter procedure, where they take the place of the variables (formal parameters) x and y”. Several good candidates explained the difference between passing by value and by reference, although this was not necessary to gain the marks. Part (c) elicited hardly any correct answers, although the fact that array elements are stored in consecutive locations in memory is virtually the definition of an array.

Part (d) attracted varied replies, although many did not really say very much about the way the loop works, the point of the question: many answers actually described a “repeat until” construction which is quite different in important respects. The important points needed for the four marks were condition test and selection, what happened in the two possible cases, and repetition. The commonest failing was no mention of what happened when the condition failed (ie character 13 entered): “ the loop stops” was common, but “ a jump to the instruction following endwhile” wasn’t. Part (e) was well done. The last part gave a mark to most candidates who gave it any thought at all, although “it would crash” without explanation was inadequate. The answer is actually dependent on the language of the program: most compilers will generate code that would produce an array-bounds execution error, although some (notoriously C or C++), would cheerfully allow the last ten characters to overflow the allotted memory and overwrite some other, potentially vital, data, causing a possible crash later.

Q27.

The majority of candidates had problems tracing this recursive algorithm. Many failed to trace the main program at all. Very few candidates indicated what each line of the trace showed making it difficult to give credit for success in some sections.

Candidates should be encouraged to show the trace as a table rather than as a rewritten algorithm with values included. In this algorithm the identifiers piece and x are local to the procedure and therefore separate instances are needed for each recursive call. The parameters a and outstring are passed by reference so there is no need to create extra columns for these as the identifiers message and newstring will be used throughout. Candidates who chose to trace a and outstring were given credit but were expected to be consistent in their use of this approach.

Most candidates scored the marks available for the first procedure call and for the correct outputs.

Q28.

A minority of weaker candidates made no attempt to answer this question. In part (a) the general principles of masking to read or set individual bits without changing others seemed not to be known by many candidates. Even when the masks were correct the logical operators were sometimes wrong.

Masks were not actually required for the algorithm in part (b), credit was given for simply stating the bits to be tested and the action needed. Where masks were given they were credited. Very few candidates bothered to check that the system was active. The vast majority neglected to provide any kind of loop to allow for continuous monitoring. Another common error was to use an ELSE clause in a way that resulted in testing the internal motion sensor and door and window contacts only when the security light had been activated. Others required both the door and window contacts to be broken before activating the alarm. Credit was given for appropriate use of a loop and IF THEN (ELSE) ENDIF constructs within the algorithm.

Answers tended to suffer where candidates failed to indent correctly. Where identifiers are

used they should be explained. Many candidates might have gained more marks if they had added comments to explain what they were trying to do.

Q29.

This question was answered successfully by the better candidates with many scoring full marks. Surprisingly, several candidates identified an internal node as a leaf node and others could not indicate a branch, clearly. In the latter case, candidates indicated their uncertainty by writing “branch” level with an internal node and omitting to link it to the tree diagram by arrow. The same identification problem arose with labelling the left and right sub-trees. Marks cannot be given when there is doubt in the mind of the examiner as to whether the candidate really knows the correct answer. The left and right sub-trees should be ringed and clearly labelled to eliminate doubt in the mind of the examiner.

Recursion has been examined frequently in recent years. It is therefore pleasing to note an increase in the number of candidates who can define the term accurately and who are able to explain why a stack is needed. Sadly, many candidates still have difficulty dry-running the execution of a recursively-defined procedure successfully. Many candidates completed the table correctly but a significant number made no attempt to show the printed output and many others produced output that was wrong. Many candidates looked at the first parameter of each call and incorrectly used it to generate the printed output. These candidates described the procedure as a *pre-order* tree traversal algorithm, whereas, in fact, it was an *in-order* traversal. Candidates who dry-ran the procedure successfully had little difficulty in stating in-order. Some candidates recognised in the layout of the procedure the structure of an *in-order* traversal. These candidates gained credit for their knowledge even though some did not complete the dry run successfully or at all.