

QUESTION 7.



- 5 (a) (i) • Set of data items have a common name (1 mark)
• Items are referenced using a subscript/index (1 mark)
• Accept: all data items are of the same data type (1 mark) [max 2]
- (ii) 24 [1]
- (iii) • The total number of amplifiers 'produced' by workers 1, 2 and 3/three workers (1 mark)
• on day 2_ (1 mark) [2]



(b)

WorkerNum	DayNum	WorkerAverage	OUTPUT
1			
2			
3			
1	1		
	2		
	3		
	4		
2	1		
	2		
	3		
	4		
3	1		
	2		
	3		
	4		
1		2.25	
2		2	
3		1.25	INVESTIGATE 3

Worker

1	2
0	
	0
	0
10	
21	
31	
45	
	20
	36
	60
	80
	9
	20
	33
	50

[8]



- (c) (i) WorkerNum : INTEGER (1 mark)
DayNum : INTEGER (1 mark)
WorkerTotal : ARRAY OF INTEGER
(1 mark) (1 mark)
WorkerAverage : REAL (1 mark) [ma]

(ii) PROCEDURE AnalyseProductionData (NumDays : INTEGER, NumWorkers :
INTEGER)

FOR WorkerNum ← 1 TO 3
WorkerTotal [WorkerNum] ← 0
ENDFOR

FOR WorkerNum ← 1 TO 3
FOR DayNum ← 1 TO 4
WorkerTotal [WorkerNum] ← WorkerTotal [WorkerNum] +
ProductionData [WorkerNum, DayNum]

ENDFOR
ENDFOR

FOR WorkerNum ← 1 TO 3
WorkerAverage = WorkerTotal [WorkerNum] / (4)*
DailyHoursWorked [WorkerNum]
IF WorkerAverage < 2
THEN
OUTPUT "Investigate" WorkerNum
ENDIF
ENDFOR

ENDPROCEDURE

Mark as follows:

All '3's changed to NumWorkers

All '4's changed to NumDays

WorkerAverage '4' changed to NumDays

[3]

(iii) (CALL) AnalyseProductionData (7, 13)

[1]

QUESTION 8.

5 (a) (i)

N	i	j	Temp	1	2	3	4	5
				11	16	13	7	8
5								
	1	1						
		2			13	16		
		3				7	16	
		4					8	16
		5						
	2	1						
		2			7	13		
		3				8	13	
		4						
	3	1		7	11			
		2			8	11		
		3						
	4	1		<i>final values are:</i>				
		2		7	8	11	13	16

Terminates at 5 and 4 respectively

Must be:

- Preceded by other entries ...
- nothing after the 1, 2 sequence

Note these final values will not be shown on the same row ...

[8]

(ii) To sort / to order/put in ascending order the items (in the array)

[1]

(iii) There were no swaps on the last pass / on pass 4

[1]

Page 7	Mark Scheme	
	Cambridge International AS/A Level – October/November 2015	



(b)

Identifier	Data Type	Description
Num		
N	INTEGER	The number of numbers in the list
i	INTEGER	Loop counter // The number of 'passes' up through the list
j	INTEGER	The <u>index</u> // position in the array
Temp	INTEGER	Description must imply/states the 'swapping' operation

Mark as follows:

INTEGER × 4

One mark per description

[1]
[4]

QUESTION 9.



3 (b)(ii): Python

```
StartPos = int(input("Enter start position: "))
NumToChange = int(input("Enter number to change: "))
for n in range(NumToChange) :
    NewChar = input("Input new value for position: ")
    LookUp[StartPos + n - 1] = NewChar
print(str(NumToChange) + " entries changed")
```

ALTERNATIVE:

```
StartPos = int(input("Enter start position: "))
NumToChange = int(input("Enter number to change: "))
n = 0
while n < NumToChange :
    NewChar = input("Input new value for position: ")
    LookUp[StartPos + n] = NewChar
    n = n + 1
print(str(NumToChange) + " entries changed")
```



5 (b): VB.NET

A StreamWriter() solution:

```
Sub InputData()
    Dim CDTitle, CDArtist, CDLocation As String
    Dim FileHandle As IO.StreamWriter
    FileHandle = New IO.StreamWriter("MyMusic.txt") ("MyMusic.txt")
    Console.WriteLine("Input CD Title: ")
    CDTitle = Console.ReadLine()
    Do Until CDTitle = "##"
        Console.WriteLine("Input CD Artist: ")
        CDArtist = Console.ReadLine()
        Console.WriteLine("Input CD Location: ")
        CDLocation = Console.ReadLine()
        FileHandle.WriteLine(CDTitle & ":" & CDArtist & ":" & CDLocation)
        Console.WriteLine("Input CD Title: ")
        CDTitle = Console.ReadLine()
    Loop
    FileHandle.Close()
End Sub
```

A legacy FileOpen() solution:

```
Sub InputData()
    Dim CDTitle, CDArtist, CDLocation As String
    FileOpen(1, "MyMusic", OpenMode.Output)
    Console.WriteLine("Input CD Title: ")
    CDTitle = Console.ReadLine()
    Do Until CDTitle = "##"
        Console.WriteLine("Input CD Artist: ")
        CDArtist = Console.ReadLine()
        Console.WriteLine("Input CD Location: ")
        CDLocation = Console.ReadLine()
        Print(1, CDTitle & ":" & CDArtist & ":" & CDLocation)
        Console.WriteLine("Input CD Title: ")
        CDTitle = Console.ReadLine()
    Loop
    FileClose(1)
End Sub
```

Page 14	Mark Scheme	
	Cambridge International AS/A Level – May/June 2016	



5 (b): Pascal

```

procedure InputData;
var
    CDTitle, CDArtist, CDLocation : string;
    CDFile : Textfile;
begin
    assign(CDFile, 'MyMusic');
    rewrite(CDFile);
    writeln('Input CD Title: ');
    readln(CDTitle);
    while (CDTitle <> '##') do
    begin
        writeln('Input CD Artist: ');
        readln(CDArtist);
        writeln('Input CD Location: ');
        readln(CDLocation);
        writeln(CDFile, CDTitle + ':' + CDArtist + ':' + CDLocation);
        writeln('Input CD Title: ');
        readln(CDTitle);
    end;
    close(CDFile);
end;

```

5 (b): Python

```

def InputData() :
    #CDTitle String (or CDTitle = "")
    #CDArtist String (or CDArtist = "")
    #CDLocation String (or CDLocation = "")

    FileHandle = open("MyMusic", "w")
    CDTitle = input("Input CD Title: ")
    while CDTitle != "##" :
        CDArtist = input("Input CD Artist: ")
        CDLocation = input("Input CD location: ")
        FileHandle.write(CDTitle + ":" + CDArtist + ":" + CDLocation)
        CDTitle = input("Input CD Title: ")
    FileHandle.close()

```

QUESTION 10.



3 (a)

[11]

```
FUNCTION Decrypt (Lookup : ARRAY, CipherChar : CHAR) RETURNS CHAR
  DECLARE Found           : BOOLEAN
  DECLARE Index          : INTEGER
  DECLARE OriginalChar    : CHAR

  Index ← 1                //start with first element in the array //
                           assign the start index

  Found ← FALSE
  //now search for CipherChar in Lookup:
  WHILE Found = FALSE // Found <> TRUE // NOT Found
    // compare CipherChar with this array element:
    IF Lookup[Index] = CipherChar
      THEN
        Found ← TRUE           // Set the flag
      ELSE
        Index ← Index + 1    //Move to next array element
      ENDIF
    ENDWHILE
  //dropped out of loop so must have found CipherChar:
  OriginalChar ← CHR(Index) // convert Index to original character
  RETURN OriginalChar

ENDFUNCTION
```

One mark for each part-statement (shown underlined and bold)

Page 5	Mark Scheme	
	Cambridge International AS/A Level – May/June 2016	



(b) 'Pseudocode' solution included here for development and clarification of scheme. Programming language example solutions appear in the Appendix.

```

INPUT StartIndex
INPUT NumberToOutput } ①

FOR Index ← StartIndex to StartIndex + NumberToOutput - 1
    OriginalChar ← CHR(Index) ③
    CipherChar ← Lookup[Index] ④
    OUTPUT ("Index " & Index & ": Character " & OriginalChar & ⑤
           " has substitute character " & CipherChar) ⑥
ENDFOR
  
```

② {

Mark points as circled, descriptions as below:

1. Two INPUT statements
2. Working loop using Index (allow alternative solutions including separate loop counter)
3. Assignment (using correct values of Index or other variable)
4. Assignment (using correct values of Index or other variable)
5. One mark for OUTPUT of a string combining text and variables

QUESTION 11.



end.

4(e): Python

```
def ProductCodeSearch(SearchCode):  
    # list indexes start at zero  
    i = 0  
    Found = "no"  
    while not(i == 1001 or Found == "yes"):  
        if SearchCode == PCode[i]:  
            Found = "yes"
```

QUESTION 12.



- 5 (a) • 2D array
• of type integer
• with identifier `PlayerScore`

[Max. 2]

- (b) (i) Stepwise refinement // Top-Down Design

[1]

- (ii) 'Pseudocode' solution included here for development and clarification of mark scheme. Programming language example solutions appear in the Appendix.

```
DECLARE ThisPlayerName : STRING
DECLARE PlayerName : ARRAY[1:8) OF STRING
DECLARE i : INTEGER

OPENFILE "NAMES.TXT" FOR READ
i ← 1

WHILE NOT EOF("NAMES.TXT")
    READFILE "NAMES.TXT", ThisPlayerName
    PlayerName[i] ← ThisPlayerName
    i ← i + 1
ENDWHILE

CLOSEFILE "NAMES.TXT"
```

One mark for each of:

- File open in read mode
- Loop until `EOF()` or count-controlled (8 iterations)
- Read a line from the file in a loop
- Assignment to `PlayerName[1 to 8]` from the file in a loop

Close file

[Max. 4]



- (iii) 'Pseudocode' solution included here for development and clarification of marks. Programming language example solutions appear in the Appendix.

```
// search for player name ....
Found ← FALSE
i ← 1

REPEAT
  IF ThisPlayerName = PlayerName[i]
    THEN
      Found ← TRUE
      PlayerNumber ← i
    ELSE
      i ← i + 1
  ENDIF

UNTIL (Found = TRUE) OR (i = 9)
```

One mark for each of:

- Initialise `i` to 1 and Found to FALSE
- Loop through array `PlayerName` (including exit when found)
- Comparison: `ThisPlayerName = PlayerName[i]` in a loop
- Found set to TRUE if `ThisPlayerName` found

[Max. 4]

- (c) (i) • a nested // an inner and an outer
• count controlled // incremented loop(s)

[2]

(ii)

...	True
...	False

Both answers must be correct

[1]

- (iii) Error line number 5, 9 or 11 as follows:

Line 5:

The boundary value must be included //

`IF PlayerScore[GameIndex, PlayerIndex] >= 100 // > 99`

Line 9:

The boundary value must be included //

`IF PlayerScore[GameIndex, PlayerIndex] >= 50 // > 49`

Line 11:

One should be added to `Total50` (not `GameIndex`) //

`Total50 ← Total50 + 1`

One mark for line number + explanation

[1]

QUESTION 13.



Question	Answer
5	Pseudocode solution included here for development and clarification of mark scheme. Programming language example solutions appear in the Appendix. <pre> PROCEDURE SearchFile() DECLARE FileData : STRING DECLARE MyArrayRow : INTEGER DECLARE SearchID : STRING MyArrayRow ← 0 / 1 OPEN "Loginfile.txt" FOR READ INPUT SearchID WHILE NOT EOF("Loginfile.txt") READFILE "Loginfile.txt", Filedata IF SearchID = LEFT(FileData,5) THEN LoginEvents[MyArrayRow,1] ← MID(Filedata, 6, 4) LoginEvents[MyArrayRow,2] ← RIGHT(Filedata, 14) MyArrayRow ← MyArrayRow + 1 ENDIF ENDWHILE CLOSEFILE("LoginFile.txt") ENDPROCEDURE </pre> 1 mark for each of the following: 1. Procedure heading and ending 2. Declare MyArrayRow as integer // commented in python 3. Initialising MyArrayRow 4. Input SearchID 5. Open file "LoginFile.txt" for input / read 6. Correct loop incorporating EOF() 7. Read a line from the file in a loop 8. Compare SearchID with correct data from file in a loop 9. Assign both values to LoginEvents[MyArray] in a loop 10. Increment MyArrayRow correctly in a loop 11. Close the file not in a loop

QUESTION 14.

Cambridge
International
AS & A Level

Cambridge International Examinations
Cambridge International Advanced Subsidiary and Advanced Level



COMPUTER SCIENCE

9608/22

Paper 2 Written Paper

May/June 2017

MARK SCHEME

Maximum Mark: 75

Published

This mark scheme is published as an aid to teachers and candidates, to indicate the requirements of the examination. It shows the basis on which Examiners were instructed to award marks. It does not indicate the details of the discussions that took place at an Examiners' meeting before marking began, which would have considered the acceptability of alternative answers.

Mark schemes should be read in conjunction with the question paper and the Principal Examiner Report for Teachers.

Cambridge will not enter into discussions about these mark schemes.

Cambridge is publishing the mark schemes for the May/June 2017 series for most Cambridge IGCSE[®], Cambridge International A and AS Level and Cambridge Pre-U components, and some Cambridge O Level components.

© IGCSE is a registered trademark.

This document consists of **8** printed pages.





Question	Answer																													
1(a)	<table><tr><th>Item</th><th>Statement</th><th>Input</th><th>Process</th><th>Output</th></tr><tr><td>1</td><td>SomeChars = "Hello World"</td><td></td><td>✓</td><td></td></tr><tr><td>2</td><td>OUTPUT RIGHT(String1,5)</td><td></td><td>✓</td><td>✓</td></tr><tr><td>3</td><td>READFILE (MyFile, String2)</td><td>✓</td><td></td><td></td></tr><tr><td>4</td><td>WRITEFILE (MyFile, "Data is " & String2)</td><td></td><td>✓</td><td>✓</td></tr></table>					Item	Statement	Input	Process	Output	1	SomeChars = "Hello World"		✓		2	OUTPUT RIGHT(String1,5)		✓	✓	3	READFILE (MyFile, String2)	✓			4	WRITEFILE (MyFile, "Data is " & String2)		✓	✓
	Item	Statement	Input	Process	Output																									
	1	SomeChars = "Hello World"		✓																										
	2	OUTPUT RIGHT(String1,5)		✓	✓																									
	3	READFILE (MyFile, String2)	✓																											
4	WRITEFILE (MyFile, "Data is " & String2)		✓	✓																										
Mark as follows: Row 1 as shown Row 2 no marks if tick in Input column, otherwise 1 mark per tick Row 3 as shown Row 4 no marks if tick in Input column, otherwise 1 mark per tick																														
1(b)(i)	- Integer / Real / Single / Double / Floating Point / Float - Boolean				2																									
1(b)(ii)	<table><tr><th>Expression</th><th>Evaluates to</th></tr><tr><td>(FlagA AND FlagB) OR FlagC</td><td>TRUE</td></tr><tr><td>FlagA AND (FlagB OR FlagC)</td><td>TRUE</td></tr><tr><td>(NOT FlagA) OR (NOT FlagC)</td><td>FALSE</td></tr></table> 1 mark per answer				Expression	Evaluates to	(FlagA AND FlagB) OR FlagC	TRUE	FlagA AND (FlagB OR FlagC)	TRUE	(NOT FlagA) OR (NOT FlagC)	FALSE	3																	
Expression	Evaluates to																													
(FlagA AND FlagB) OR FlagC	TRUE																													
FlagA AND (FlagB OR FlagC)	TRUE																													
(NOT FlagA) OR (NOT FlagC)	FALSE																													
1(c)	MyCount ← 101 REPEAT OUTPUT MyCount MyCount ← MyCount + 2 UNTIL MyCount > 199 1 mark for each of the following: - Counter initialisation - Repeat ... Until loop - Method for choosing (correct range of) odd numbers - Output all odd numbers in the range Note: Counter variable name must be consistent				4																									



Question	Answer	
2(a)	- to <u>increase the level of detail of an algorithm / design...</u> // breaking down a <u>problem / module / task</u> into smaller parts... ...from which the task may be <u>programmed</u> 1 mark per underlined phrase or equivalent	
2(b)	1 mark for first 3 data types – String 1 mark for last data type – Boolean 1 mark for each description: FileUserID Stores (User) ID from <u>file</u> FilePreferredName Stores (preferred) name from <u>file</u> IDFoundFlag True if (User) ID found in <u>file</u> // False if (User) ID not found in <u>file</u> // If SearchUserID matches FileUserID	5
2(c)	- LOOP through the file until EOF()... - OR SearchUserId is found - READ text line from UserNames.txt file in a loop - EXTRACT FileUserID in a loop - IF SearchUserId matches FileUserID THEN in a loop - SET FilePreferredName to the name from the file - Check if User ID found not in a loop - OUTPUT appropriate message for both conditions 1 mark per functional equivalent of each numbered statement.	Max 8



Question	Answer
3	<pre> FUNCTION ExCamel (<u>InString</u>: <u>STRING</u>) RETURNS <u>STRING</u> DECLARE NextChar : <u>CHAR</u> DECLARE <u>OutString</u> : <u>STRING</u> DECLARE n : <u>INTEGER</u> <u>OutString</u> ← "" // initialise the return string // loop through InString to produce OutString FOR n ← 1 TO <u>LENGTH(InString)</u> // from first to last NextChar ← <u>MID(InString, n, 1)</u> // get next character IF <u>NextChar >= 'A' AND NextChar <= 'Z'</u> // check if upper case // <u>NextChar = UCASE(NextChar)</u> THEN IF n > 1 // if not first character THEN <u>OutString</u> ← <u>OutString & " "</u> // add space to OutString ENDIF <u>NextChar</u> ← <u>LCASE(NextChar)</u> // make NextChar lower case ENDIF <u>OutString</u> ← <u>OutString & NextChar</u> // add Nextchar to OutString ENDFOR <u>RETURN OutString</u> // return value ENDFUNCTION </pre> 1 mark per underlined word / expression



Question	Answer											
4(a)	• Functions • Procedures • Global / Local variables 1 mark per item											
4(b)	<table><tr><th>Name of parameter passing method</th><th>Value output</th><th>Explanation</th></tr><tr><td>(Call) by reference</td><td>5</td><td> • The <u>address of</u> the variable is passed. • <u>Original value is changed</u> when parameter changed in called module. </td></tr><tr><td>(Call) by value</td><td>4</td><td> • A <u>copy of</u> the variable itself is passed. • <u>Original value not changed</u> when parameter changed in called module. </td></tr></table> Mark as follows: • 1 mark for each name and value • 1 mark per bullet in explanation			Name of parameter passing method	Value output	Explanation	(Call) by reference	5	• The <u>address of</u> the variable is passed. • <u>Original value is changed</u> when parameter changed in called module.	(Call) by value	4	• A <u>copy of</u> the variable itself is passed. • <u>Original value not changed</u> when parameter changed in called module.
Name of parameter passing method	Value output	Explanation										
(Call) by reference	5	• The <u>address of</u> the variable is passed. • <u>Original value is changed</u> when parameter changed in called module.										
(Call) by value	4	• A <u>copy of</u> the variable itself is passed. • <u>Original value not changed</u> when parameter changed in called module.										



6

Question	Answer	Marks
5(a)(i)	• Any character <u>except</u> colon, space or any alpha-numeric • Reason: character is not in the login information strings	2
5(a)(ii)	DECLARE <u>LogArray</u> : ARRAY[1 : 20] OF <u>STRING</u> 1 mark per underline	2



Question	Answer
5(b)	Pseudocode solution included here for development and clarification of mark scheme. Programming language example solutions appear in the Appendix. <pre> PROCEDURE LogEvents() DECLARE FileData : STRING DECLARE ArrayIndex : INTEGER OPENFILE "LoginFile.txt" FOR APPEND FOR ArrayIndex ← 1 TO 20 // IF LogArray[ArrayIndex] <> "*****" THEN FileData ← LogArray[ArrayIndex] WRITEFILE ("LoginFile.txt", FileData) ENDIF ENDFOR CLOSEFILE("LoginFile.txt") ENDPROCEDURE </pre> 1 mark for each of the following: 1. Procedure heading and ending 2. Declare <code>ArrayIndex</code> as integer // commented in python 3. Open file 'LoginFile' for append 4. Correct loop 5. extract data from array in a loop 6. check for unused element in a loop 7. write data to file in a loop 8. Close the file outside the loop



Question	Answer
6(a)	Pseudocode solution included here for development and clarification of mark scheme. Programming language example solutions appear in the Appendix. <pre> FUNCTION ValidateRegistration(Registration : STRING) RETURNS BOOLEAN DECLARE UCaseChar, NumChar : INTEGER DECLARE NextChar : CHAR DECLARE ReturnFlag : BOOLEAN DECLARE n : INTEGER ReturnFlag ← TRUE ValidateRegistration ← True IF LEN(Registration) < 6 OR LEN(Registration) > 9 //check length THEN ReturnFlag ← False ELSE FOR n ← 1 TO 3 //check for 3 upper case alpha NextChar ← MID(Registration, n, 1) IF NextChar < 'A' AND NextChar > 'Z' THEN ReturnFlag ← False ENDIF ENDFOR FOR n ← 4 TO 5 //check for 2 numeric NextChar ← MID(Registration, n, 1) IF NextChar < '0' AND NextChar > '9' THEN ReturnFlag ← False ENDIF ENDFOR FOR n ← 6 TO LEN(Registration) //check remaining characters NextChar ← MID(Registration, n, 1) IF NextChar < 'A' AND NextChar > 'Z' THEN ReturnFlag ← False ENDIF ENDFOR ENDIF RETURN (ReturnFlag) ENDFUNCTION </pre>



Question	Answer	
6(a)	1 mark for each of the following: 1. Correct Function heading and ending 2. Check for correct length 3. Extract first three characters 4. Check first three characters are capitals 5. Extract characters four and five 6. Check characters four and five are numeric 7. Extract remaining characters 8. Check remaining characters are capitals 9. Combine all four tests results into a single Boolean value 10. Return a Boolean value	
6(b)	String1: (for example, "ABC12XYZ") One mark for a valid string having: • Correct length (between 6 and 9 characters) • 3 capital letters followed by... • 2 numeric characters followed by... • between 1 and 4 capital letters String2 to String5: 1 mark for each string and explanation (testing different rules of the function) Test strings breaking one different rules: • Incorrect length • With incorrect number of capital letters at the start • With non-numeric characters in positions 4 and 5 • With incorrect number of capital letters at the end • Containing an invalid character (not alpha-numeric)	5

QUESTION 15.

Cambridge
International
AS & A Level

Cambridge Assessment International Education
Cambridge International Advanced Subsidiary and Advanced Level



COMPUTER SCIENCE

9608/21

Paper 2 Written Paper

October/November 2017

MARK SCHEME

Maximum Mark: 75

Published

This mark scheme is published as an aid to teachers and candidates, to indicate the requirements of the examination. It shows the basis on which Examiners were instructed to award marks. It does not indicate the details of the discussions that took place at an Examiners' meeting before marking began, which would have considered the acceptability of alternative answers.

Mark schemes should be read in conjunction with the question paper and the Principal Examiner Report for Teachers.

Cambridge International will not enter into discussions about these mark schemes.

Cambridge International is publishing the mark schemes for the October/November 2017 series for most Cambridge IGCSE[®], Cambridge International A and AS Level components and some Cambridge O Level components.

© IGCSE is a registered trademark.

This document consists of **12** printed pages.

 **Cambridge Assessment**
International Education



Question	Answer															
1(a)(i)	<table><tr><th>Data value</th><th>Data type</th></tr><tr><td>27</td><td>INTEGER</td></tr><tr><td>"27"</td><td>STRING</td></tr><tr><td>"27.3"</td><td>STRING</td></tr><tr><td>TRUE</td><td>BOOLEAN</td></tr><tr><td>27/3/2015</td><td>DATE // DATETIME</td></tr><tr><td>27.3</td><td>REAL</td></tr></table> One mark for each data type Mark first data type given in each case	Data value	Data type	27	INTEGER	"27"	STRING	"27.3"	STRING	TRUE	BOOLEAN	27/3/2015	DATE // DATETIME	27.3	REAL	
Data value	Data type															
27	INTEGER															
"27"	STRING															
"27.3"	STRING															
TRUE	BOOLEAN															
27/3/2015	DATE // DATETIME															
27.3	REAL															
1(a)(ii)	1D Array // 1DList	2														
1(a)(iii)	- Each character is represented by an <u>unique</u> / <u>corresponding</u> - binary code / integer / value	2														
1(b)	- When a section of code would be repeated - When a piece of code is needed to perform a specific task - To support modular programming / step wise refinement - Easier to debug / maintain - Built-in / library routines are tried and tested One mark per answer	Max 2														
1(c)	<pre>CASE OF MyVar 1: CALL Proc1() 2: CALL Proc2() 3: CALL Proc3() OTHERWISE OUTPUT "Error" ENDCASE</pre> One mark for: - First line and ENDCASE - All clauses for 1, 2 and 3 'OTHERWISE' clause - OUTPUT statement	4														



Question	Answer
1(d)	Ability to recognise: • selection statement • iteration statement • assignment statements • data declarations / structures / data types / use of variables or objects • modular structure / functions / procedures / subroutines • subroutine parameters • Specific types of statement, e.g. Input, Output, File operations • Code format • Operators Mark as follows: Any two from above, or valid alternative Accept by example

Question	Answer	Marks																																										
2(a)	<table><tr><th>StartNumber</th><th>EndNumber</th><th>Divisor</th><th>NumberFound</th><th>Number</th><th>Remainder</th><th>Output</th></tr><tr><td>11</td><td>13</td><td>2</td><td>0</td><td>11</td><td>1</td><td></td></tr><tr><td></td><td></td><td></td><td></td><td>12</td><td>0</td><td>12</td></tr><tr><td></td><td></td><td></td><td>1</td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td>13</td><td>1</td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td>Count: 1</td></tr></table> One mark for correct Number column One mark for correct Remainder column One mark for correct Output	StartNumber	EndNumber	Divisor	NumberFound	Number	Remainder	Output	11	13	2	0	11	1						12	0	12				1								13	1								Count: 1	3
StartNumber	EndNumber	Divisor	NumberFound	Number	Remainder	Output																																						
11	13	2	0	11	1																																							
				12	0	12																																						
			1																																									
				13	1																																							
						Count: 1																																						
2(b)	Mark as follows: - For a (given) range of values - Counts the number of times one number (numerator) is an exact divisor of the other - Outputs each numerator (only) - Outputs the count Accept by example No mark for: ...calculate the remainder ...add one to NumberFound	3																																										



Question	Answer
2(c)	<pre> graph TD Start([START]) --> Input[/INPUT StartNumber, EndNumber, Divisor/] Input --> Init1[NumberFound ← 0] Init1 --> Init2[Number ← StartNumber] Init2 --> LoopStart(()) LoopStart --> Mod[Remainder ← MODULUS(Number, Divisor)] Mod --> Div{Remainder = 0?} Div -- YES --> Out1[/OUTPUT Number/] Out1 --> IncFound[Increment NumberFound] IncFound --> LoopStart Div -- NO --> LoopStart LoopStart --> EndCheck{Number = EndNumber?} EndCheck -- YES --> Out2[/OUTPUT "Count: " & NumberFound/] Out2 --> Stop([STOP]) EndCheck -- NO --> IncNum[Increment Number] IncNum --> LoopStart </pre> The flowchart describes a process to count the number of integers between StartNumber and EndNumber (inclusive) that are divisible by Divisor. It begins with a START terminal, followed by an input block for StartNumber, EndNumber, and Divisor. Two initialization steps are shown: NumberFound ← 0 and Number ← StartNumber, which are grouped by a bracket. The process then enters a loop. The first step in the loop is to calculate the remainder of Number divided by Divisor using the MODULUS function. A decision diamond checks if the remainder is 0. If YES, the current Number is output, and NumberFound is incremented. If NO, the process skips the output and increment steps. Both paths lead to a second decision diamond: Number = EndNumber?. If YES, the final count is output as "Count: " followed by NumberFound, and the process stops at the STOP terminal. If NO, the Number is incremented, and the loop repeats from the MODULUS step.



Question	Answer
2(c)	Mark as follows: • One mark for <code>START</code> and <code>STOP / END</code> • One mark for bracketed pair • One mark for each of other labelled boxes (shape must be correct for decision box) Decision box outputs must have two outputs and at least one label (<code>Yes / No</code>) Different statement categories should not appear in the same symbol (e.g. assignment and I/O) No mark for symbol (or pair) if parent missing or logically incorrect (except for <code>START/END</code>) Full marks should be awarded for functionally equivalent solutions.



Question	Answer
3(a)	<pre> PROCEDURE BubbleSort DECLARE Temp : STRING DECLARE FirstID, SecondID : INTEGER DECLARE NoSwaps : BOOLEAN DECLARE Boundary : INTEGER Declare J : INTEGER Boundary ← 99 REPEAT NoSwaps ← TRUE FOR J ← 1 TO Boundary FirstID ← UserNameArray[J] SecondID ← UserNameArray[J + 1] IF FirstID > SecondID THEN Temp ← UserNameArray[J] UserNameArray[J] ← UserNameArray[J + 1] UserNameArray[J + 1] ← Temp NoSwaps ← FALSE ENDIF ENDFOR Boundary ← Boundary - 1 UNTIL NoSwaps = TRUE ENDPROCEDURE </pre> Mark as follows: 1. Procedure heading and ending (allow array as input parameter) 2. Variable declaration for counter / index (integer) or temp (string) 3. Outer working loop 4. Inner loop with suitable range 5. Correct comparison in a loop 6. Correct swap of complete array element in a loop 7. Set flag to indicate swap in inner loop and resetting in outer loop 8. Reducing Boundary in a loop



Question	Answer
3(b)	Pseudocode solution included here for development and clarification of mark scheme. Programming language example solutions appear in the Appendix. <pre> PROCEDURE FindRepeats DECLARE i, RepeatCount: INTEGER DECLARE FirstID, SecondID: STRING RepeatCount ← 0 FOR i ← 2 TO 100 FirstID ← LEFT(UsernameArray[i - 1], 6) SecondID ← LEFT(UsernameArray[i], 6) IF FirstID = SecondID THEN RepeatCount ← RepeatCount + 1 OUTPUT(UsernameArray[i]) ENDIF ENDFOR IF RepeatCount = 0 THEN OUTPUT "The array contains no repeated UserIDs" ELSE OUTPUT "There are " & RepeatCount & " repeated userIDs" ENDIF ENDPROCEDURE </pre> Mark as follows (all must be correct syntax for chosen language): 1. Procedure heading and ending 2. Variable declaration for INTEGER (comment in Python) and initialisation for RepeatCount (or equivalent name) 3. Loop 4. Extraction of UserID in a loop 5. Correct comparison of consecutive elements... in a loop 6. ...output correct array element (NOT original, only duplicates) in a loop 7. increment RepeatCount following a comparison in a loop 8. Correct conditional statement checking RepeatCount (or equivalent) and then two correct final OUTPUT statements



Question	Answer	
3(c)(i)	• Problem definition • Design • Coding / programming • Testing • Documentation • Implementation • Maintenance	
3(c)(ii)	<u>Integrated Development Environment</u> or a suitable description	1
3(c)(iii)	Examples include: • context sensitive prompts • (dynamic) syntax checking • use of colours to highlight key words / pretty printing • Formatting • Single-stepping • Breakpoints • Report / watch window • (UML) modelling • Compiler/interpreter • Text editor	Max 2
3(c)(iv)	Run-time	1

Question	Answer	Marks												
4(a)	<table><thead><tr><th>Value</th><th>Formatted String</th></tr></thead><tbody><tr><td>1327.5</td><td>"□ 1327.50"</td></tr><tr><td>1234</td><td>"□ 1234.00"</td></tr><tr><td>7.456</td><td>"□□□ 07.45"</td></tr></tbody></table> Leading spaces must be present	Value	Formatted String	1327.5	"□ 1327.50"	1234	"□ 1234.00"	7.456	"□□□ 07.45"	2				
Value	Formatted String													
1327.5	"□ 1327.50"													
1234	"□ 1234.00"													
7.456	"□□□ 07.45"													
4(b)	<table><thead><tr><th>Value</th><th>Required output</th><th>Mask</th></tr></thead><tbody><tr><td>1234.00</td><td>"1,234.00"</td><td>"0,000.00"</td></tr><tr><td>3445.66</td><td>"£3,445.66"</td><td>"£0,000.00"</td></tr><tr><td>10345.56</td><td>"\$□□10,345"</td><td>"\$##00,000"</td></tr></tbody></table> Currency and 'punctuation' symbols must be as shown	Value	Required output	Mask	1234.00	"1,234.00"	"0,000.00"	3445.66	"£3,445.66"	"£0,000.00"	10345.56	"\$□□10,345"	"\$##00,000"	3
Value	Required output	Mask												
1234.00	"1,234.00"	"0,000.00"												
3445.66	"£3,445.66"	"£0,000.00"												
10345.56	"\$□□10,345"	"\$##00,000"												



Question	Answer	
5(a)	<pre> PROCEDURE MakeNewfile DECLARE OldFileLine : STRING DECLARE NewFileLine : STRING OPENFILE "EmailDetails" FOR READ OPENFILE "NewEmailDetails" FOR WRITE WHILE NOT EOF("EmailDetails") READFILE "EmailDetails", OldFileLine NewFileLine ← "00" & OldFileLine WRITEFILE "NewEmailDetails", NewFileLine ENDWHILE CLOSEFILE "EmailDetails" CLOSEFILE "NewEmailDetails" ENDPROCEDURE </pre> Mark as follows: 1. Variable declaration of <code>STRING</code> for <code>OldFileLine</code> (or equivalent) 2. Open <code>EmailDetails</code> for <code>READ</code> 3. Open <code>NewEmailDetails</code> for <code>WRITE</code> 4. Correct loop checking for <code>EOF (EmailDetails)</code> 5. Reading a line from <code>EmailDetails</code> in a loop 6. Correct concatenation in a loop 7. Writing a line to <code>NewEmailDetails</code> in a loop Closing both files	
5(b)	Invalid string examples: A string with nothing before '@' A string with nothing after '@' A string with 1 or 2 characters after '@' A string with no '@' symbol A string with more than one '@' symbol Explanation Sensible explanation mapping each given string to an individual rule One mark for string One mark for explanation Each rule should be tested once only	6

**Programming Example Solutions****Q3(b): Visual Basic**

```
Sub FindRepeats()  
    Dim Repeats As Integer  
    Dim i As Integer  
    Dim FirstID As String  
    Dim SecondID As String  
  
    Repeats = 0  
    For i = 1 To 99  
        FirstID = Left(UsernameArray(i), 6)  
        SecondID = Left(UsernameArray(i + 1), 6)  
        If FirstID = SecondID Then  
            Console.WriteLine(UsernameArray(i + 1))  
            Repeats = Repeats + 1  
        End If  
    Next i  
  
    If Repeats = 0 Then  
        Console.WriteLine("The array contains no repeated UserIDs")  
    Else  
        Console.WriteLine("There are " & Repeats & " repeated UserIDs")  
    End If  
  
End Sub
```

Alternative:

```
Sub FindRepeats ()  
  
    Dim RepeatCount, i As Integer  
    Dim FirstID, SecondID As String  
  
    RepeatCount = 0  
    For i = 1 to 99  
        FirstID = Left(UsernameArray(i-1),6)  
        SecondID = Left(UsernameArray(i),6)  
        If FirstID = SecondID then  
            Console.WriteLine (UsernameArray(i))  
            RepeatCount = RepeatCount + 1  
        End If  
    Next i  
  
    If RepeatCount = 0 then  
        Console.WriteLine ("The array contains no repeated UserIDs")  
    Else  
        Console.WriteLine ("There are "& RepeatCount & " repeated UserIDs")  
    End If  
  
End Sub
```

**Q3(b): Pascal**

```
procedure FindRepeats ();  
  
var  
    RepeatCount, i : integer;  
    FirstID, SecondID : string;  
  
begin  
    RepeatCount := 0;  
    for i := 1 to 99 do  
        begin  
            FirstID := Copy(UsernameArray[i-1],1,6);  
            SecondID := Copy(UsernameArray[i],1,6);  
            if FirstID = SecondID then  
                begin  
                    writeln (UsernameArray[i]);  
                    RepeatCount := RepeatCount + 1;  
                end;  
            end;  
  
            if RepeatCount = 0 then  
                writeln ('The array contains no repeated UserIDs')  
            else  
                writeln ('There are ', RepeatCount, ' repeated UserIDs')  
            end;  
        end;  
    end;
```

**Q3(b): Python**

```
def FindRepeats():
    #Repeats, i Integer
    #FirstID, SecondID string
    Repeats = 0
    for i in range(0, len(UserNameArray)-1):
        FirstID = (UserNameArray[i])[:6]
        SecondID = (UserNameArray[i+1])[:6]
        if FirstID == SecondID:
            print(UserNameArray[i+1])
            Repeats = Repeats + 1
    if Repeats == 0:
        print("The array contains no repeated UserIDs")
    else:
        print("There are ", Repeats, " repeated UserIDs")
```

Alternative:

```
def FindRepeats ():

    RepeatCount = 0                                ## Defined as an integer

    for i in range (1,100):                          ## depending on next two
lines(0,99) (2,101)
        FirstID = UserNameArray[i-1]                ## Defined as string
        SecondID = UserNameArray[i]                  ## Defined as string
        if FirstID[0:6] == SecondID[0:6]:            ## Using split
            print (UserNameArray[i])
            RepeatCount += 1

    if repeatCount == 0:
        print ('The array contains no repeated UserIDs')
    else:
        print ('There are ', RepeatCount, ' repeated UserIDs')
```

QUESTION 16.



Cambridge Assessment International Education
Cambridge International Advanced Subsidiary and Advanced Level



COMPUTER SCIENCE

9608/22

Paper 2 Written Paper

October/November 2017

MARK SCHEME

Maximum Mark: 75

Published

This mark scheme is published as an aid to teachers and candidates, to indicate the requirements of the examination. It shows the basis on which Examiners were instructed to award marks. It does not indicate the details of the discussions that took place at an Examiners' meeting before marking began, which would have considered the acceptability of alternative answers.

Mark schemes should be read in conjunction with the question paper and the Principal Examiner Report for Teachers.

Cambridge International will not enter into discussions about these mark schemes.

Cambridge International is publishing the mark schemes for the October/November 2017 series for most Cambridge IGCSE[®], Cambridge International A and AS Level components and some Cambridge O Level components.

© IGCSE is a registered trademark.

This document consists of **12** printed pages.



Cambridge Assessment
International Education



Question	Answer															
1(a)(i)	<table><tr><th>Data value</th><th>Data type</th></tr><tr><td>FALSE</td><td>BOOLEAN</td></tr><tr><td>03/03/2013</td><td>DATE // DATETIME</td></tr><tr><td>35</td><td>INTEGER</td></tr><tr><td>"INTEGER"</td><td>STRING</td></tr><tr><td>3.5</td><td>REAL</td></tr><tr><td>"35"</td><td>STRING</td></tr></table> One mark for each data type Mark first data type given in each case	Data value	Data type	FALSE	BOOLEAN	03/03/2013	DATE // DATETIME	35	INTEGER	"INTEGER"	STRING	3.5	REAL	"35"	STRING	
Data value	Data type															
FALSE	BOOLEAN															
03/03/2013	DATE // DATETIME															
35	INTEGER															
"INTEGER"	STRING															
3.5	REAL															
"35"	STRING															
1(a)(ii)	1D Array // 1D List	1														
1(a)(iii)	Ability to recognise: • selection statement • iteration statement • assignment statements • data declarations / structures / data types / use of variables or objects • modular structure / functions / procedures / subroutines • subroutine parameters • Specific types of statement e.g. Input, Output, File operations • Code format • Operators Mark as follows: Any two from above, or valid alternative Accept by example	2														
1(b)(i)	<table><tr><th>Data</th></tr><tr><td></td></tr><tr><td>67 // 0100 0011 // 043h</td></tr><tr><td>65 // 0100 0001 // 041h</td></tr><tr><td>71 // 0100 0111 // 047h</td></tr><tr><td>69 // 0100 0101 // 045h</td></tr><tr><td></td></tr></table> One mark for 67 and 65 One mark for 71 and 69 Accept binary, denary or hex values (hex must be clearly indicated) Max one mark if blank cell anywhere in sequence Ignore any data values before or after the four characters	Data		67 // 0100 0011 // 043h	65 // 0100 0001 // 041h	71 // 0100 0111 // 047h	69 // 0100 0101 // 045h		2							
Data																
67 // 0100 0011 // 043h																
65 // 0100 0001 // 041h																
71 // 0100 0111 // 047h																
69 // 0100 0101 // 045h																



Question	Answer	
1(b)(ii)	• A value representing the number of characters stored at beginning of string - OR • Terminator / special character stored to indicate the end of string One mark for each phrase or equivalent.	
1(c)	Explanation includes: • to pass values to/from the subroutine • to produce re-useable code • to avoid global variables • to allow recursion One mark per answer	Max 3
1(d)(i)	27: MyGrade assigned the value "Fail" 101: Output the text "Invalid Value Entered" Ignore minor spelling mistakes	2
1(d)(ii)	<pre> IF MyMark >= 75 AND MyMark <=100 THEN MyGrade ← "Distinction" ELSE IF MyMark >= 35 AND MyMark <=74 THEN MyGrade ← "Pass" ELSE IF MyMark >= 0 AND MyMark <=34 THEN MyGrade ← "Fail" ELSE OUTPUT "Invalid value entered" ENDIF ENDIF ENDIF ENDIF </pre> One mark for each of: • One correct range test • 'IF' equivalent (nested or not) to three CASE range tests... • ... with three corresponding assignments • Equivalent of CASE OTHERWISE with corresponding OUTPUT statement • Matching (three) ENDIFS (Or one if ELSIFS used) Max 4 if solution doesn't work under all circumstances // is not functionally equivalent to CASE	5



Question	Answer	
2(a)	Mark as follows: - To search for a given value in the array - and output the position in the array if the search value is found - if search value not found then output "Not found"	
2(b)	<pre> graph TD Start([START]) --> Input[/Input SearchValue/] Input --> FoundFlag[FoundFlag ← FALSE] FoundFlag --> Index[Index ← 1] Index --> LoopCond{Index < 101 AND FoundFlag = FALSE?} LoopCond -- YES --> ArrayCheck{ClassName[Index] = SearchVal?} ArrayCheck -- YES --> OutputIndex[/OUTPUT Index/] OutputIndex --> FoundFlagTrue[FoundFlag ← TRUE] FoundFlagTrue --> IncrementIndex[Increment Index] ArrayCheck -- NO --> IncrementIndex LoopCond -- NO --> FoundFlagFalse{FoundFlag = FALSE?} FoundFlagFalse -- YES --> OutputNotFound[/OUTPUT "Not found"/] FoundFlagFalse -- NO --> Stop([STOP]) IncrementIndex --> LoopCond </pre>	9



Question	Answer
2(b)	Mark as follows: - One mark for START and STOP / END - One mark for each bracketed pair - One mark for each of other labelled symbol (decision box shape must be correct) - Allow F/T from incorrect decision symbol Full marks should be awarded for functionally equivalent solutions.

Question	Answer	Marks																																	
3	<table border="1"> <thead> <tr> <th>Line number</th><th>Error</th><th>Correction</th></tr> </thead> <tbody> <tr> <td>01</td><td>Wrong procedure name – "SortArray"</td><td>PROCEDURE ArraySort</td></tr> <tr> <td>02</td><td>Wrong data type - CHAR</td><td>DECLARE Temp: STRING</td></tr> <tr> <td>03</td><td>Variables undefined</td><td>DECLARE FirstID, SecondID, I, J : INTEGER</td></tr> <tr> <td>04</td><td>Wrong 'Value2' of 100</td><td>FOR I ← 1 TO 99</td></tr> <tr> <td>05</td><td>Wrong range</td><td>FOR J ← 1 TO (100 - I)</td></tr> <tr> <td>06/07</td><td>Wrong function - MODULUS</td><td>Replace MODULUS with TONUM: FirstID ← TONUM(LEFT(Product[J],</td></tr> <tr> <td>06/07</td><td>Wrong value of 6</td><td>Should be 4: FirstID ← TONUM(LEFT(Product[J],</td></tr> <tr> <td>10</td><td>Assigning wrong value to Temp</td><td>Temp ← Product[J]</td></tr> <tr> <td>11</td><td>Assigning wrong value to Product[I]</td><td>Product[J] ← Product[J + 1]</td></tr> <tr> <td>13/14</td><td>Lines reversed</td><td>13 ENDIF 14 ENDFOR</td></tr> </tbody> </table> One mark for each correct row	Line number	Error	Correction	01	Wrong procedure name – "SortArray"	PROCEDURE ArraySort	02	Wrong data type - CHAR	DECLARE Temp: STRING	03	Variables undefined	DECLARE FirstID, SecondID, I, J : INTEGER	04	Wrong 'Value2' of 100	FOR I ← 1 TO 99	05	Wrong range	FOR J ← 1 TO (100 - I)	06/07	Wrong function - MODULUS	Replace MODULUS with TONUM: FirstID ← TONUM (LEFT(Product[J],	06/07	Wrong value of 6	Should be 4: FirstID ← TONUM(LEFT(Product[J],	10	Assigning wrong value to Temp	Temp ← Product[J]	11	Assigning wrong value to Product[I]	Product[J] ← Product[J + 1]	13/14	Lines reversed	13 ENDIF 14 ENDFOR	Max 8
Line number	Error	Correction																																	
01	Wrong procedure name – "SortArray"	PROCEDURE ArraySort																																	
02	Wrong data type - CHAR	DECLARE Temp: STRING																																	
03	Variables undefined	DECLARE FirstID, SecondID, I, J : INTEGER																																	
04	Wrong 'Value2' of 100	FOR I ← 1 TO 99																																	
05	Wrong range	FOR J ← 1 TO (100 - I)																																	
06/07	Wrong function - MODULUS	Replace MODULUS with TONUM: FirstID ← TONUM (LEFT(Product[J],																																	
06/07	Wrong value of 6	Should be 4: FirstID ← TONUM(LEFT(Product[J],																																	
10	Assigning wrong value to Temp	Temp ← Product[J]																																	
11	Assigning wrong value to Product[I]	Product[J] ← Product[J + 1]																																	
13/14	Lines reversed	13 ENDIF 14 ENDFOR																																	



Question	Answer
4(a)	Pseudocode solution included here for development and clarification of mark scheme. Programming language solutions appear in the Appendix. <pre> PROCEDURE TestRandom (Repetitions : INTEGER) DECLARE Frequency : ARRAY [1 : 10] OF INTEGER DECLARE Expected : REAL / INTEGER //allow either DECLARE NextRandom : INTEGER DECLARE N : INTEGER FOR N ← 1 TO 10 Frequency[N] ← 0 ENDFOR Expected ← INT(Repetitions / 10) CALL RANDOMIZE() //Set random seed FOR N ← 1 TO Repetitions NextRandom ← INT(RND() * 10) + 1 Frequency[NextRandom] ← Frequency[NextRandom] + 1 ENDFOR OUTPUT "The expected frequency is " & Expected OUTPUT "Number Frequency Difference" FOR N ← 1 TO 10 OUTPUT N & " " & Frequency[N] & " " & Frequency[N] - Expected ENDFOR ENDPROCEDURE </pre> Mark as follows: 1. Procedure heading (including parameter) 2. Array declaration – 10 or 11 elements 3. Array declaration – data type 4. Variable declaration for a loop counter (integer) or expected frequency (integer or real) 5. Variable declaration for next random value (For Python solutions, mark points 1 to 4 may be gained by suitable comments) 6. Initialise all elements of array 7. To set all elements to zero 8. Calculate expected frequency



Question	Answer	
4(a)	9. Loop to generate required number of random values 10. Use of relevant <code>RANDOM()</code> function in a loop 11. Generate random integer value in the range 1 to 10 in a loop 12. Increment (array) element in a loop 13. Output expected frequency message not in any loop 14. Output column header text 15. (Loop to) output each row 16. ... including three correct values (spaces optional)	
4(b)	• Single-stepping – to allow program statements to be executed one at a time • Breakpoints – to pause / stop the program at a specific line / statement • Variable / expression watch window – to monitor the value of variables / expressions as the program is run One mark for each Feature (text as above or equivalent) + 1 for meaningful explanation of use in context.	6
4(c)	Program is probably working correctly if: • Header is present giving frequency as 20 • Column headers are present • All rows are present (1 to 10) • The difference is calculated correctly • Output is formatted correctly • Total differences should be zero • Sum of Frequencies should be 200	Max 2



Question	Answer
5	<pre> PROCEDURE RemoveDetails DECLARE FileLine: STRING DECLARE MemberToDelete: STRING OPENFILE "EmailDetails.txt" FOR READ OPENFILE "NewEmailDetails.txt" FOR WRITE INPUT MembershipNumber WHILE NOT EOF("EmailDetails.txt") READFILE "EmailDetails.txt", FileLine IF LEFT(FileLine, 4) <> MembershipNumber THEN WRITEFILE "NewEmailDetails.txt", FileLine ENDIF ENDWHILE CLOSEFILE "EmailDetails.txt" CLOSEFILE "NewEmailDetails.txt" ENDPROCEDURE </pre> Mark as follows: 1. Procedure declaration and end. No parameters. 2. Variable declaration of <code>STRING</code> for variable <code>FileLine</code> (or similar) 3. Input the <code>MembershipNumber</code> of the person who has left 4. Open <code>EmailDetails</code> for <code>READ</code> 5. Open <code>NewEmailDetails</code> for <code>WRITE</code> 6. Correct loop checking for <code>EOF (EmailDetails)</code> 7. Reading a line from <code>EmailDetails.txt</code> in a loop 8. Correct check for <code>MemberToDelete</code> in a loop 9. Writing a line to <code>NewEmailDetails.txt</code> in a loop 10. Closing both files (not in a loop)



Appendix - Program Code Example Solutions

Q4 (a): Visual Basic

```

Dim random As New Random()

Sub TestRandom(ByVal repetitions As Integer)
    Dim randinrange As Integer
    Dim i As Integer
    Dim num(1 To 10) As Integer
    Dim freq As Integer
    Dim difference As Integer

    freq = repetitions / 10    'calculate expected frequency

    For i = 1 To 10            'initialise array to store total frequencies
        num(i) = 0
    Next i

    For i = 1 To repetitions 'generate random numbers & increment
        appropriate freq
        randinrange = random.Next(1, 11)
        num(randinrange) = num(randinrange) + 1
    Next i

    Console.WriteLine("The expected frequency is " & freq)    'report header
    Console.WriteLine("Number    Frequency    Difference")    'column headers

    For i = 1 To 10    'calc & display difference between expected and actual
        freq
        difference = num(i) - freq
        Console.WriteLine(i & "    " & num(i) & "    " & difference)
    Next i

End Sub

```

Other possible ways of calculating a random number in VB include:

```

    randinrange = CInt(Math.Floor((upperbound - lowerbound + 1) * Rnd())) +
lowerbound

    randinrange = math.round((Rnd()*9)+1)

    randinrange = CInt(Math.Ceiling(Rnd() * 9)

```

**Q4 (a): Pascal**

```
procedure TestRandom(var Repetitions : integer);
var
    Frequency : array[1..10] : integer;
    Expected, NextRandom, N : integer;

begin
    Expected := Round(Repetitions/10);
    for N := 1 to 10 do
        Frequency[N] := 0;

    for N := 1 to Repetitions do
        begin
            NextRandom := random(10)+1;
            Frequency[NextRandom] := Frequency[NextRandom]+1;
        end;

    writeln ('The expected frequency is ', Expected);
    writeln ('Number Frequency Difference');

    for N := 1 to 10 do
        writeln ('    ', N, '    ', Frequency[N], '    ', Frequency[N] -
Expected);

    end;
```


**Q4 (a): Python**

```
# frequency is an array from 1 to 10 of type integer;
# nextNumber is an integer which stores the created random number
# expected is an integer which stores the expected frequency of each number

def TestRandom (repetitions):
    import random
    frequency = [0 for i in range(1,11)] # initialise each frequency count to 0

    expected = repetitions / 10

    for i in range(1, repetitions + 1):
        nextNumber = random.randint(1,10)
        frequency[nextNumber] = frequency[nextNumber]+ 1

    print ("The expected frequency is ", expected)
    print("    Number    Frequency    Difference")

    for i in range(1,11):
        print ("    ", i, "    ", frequency[i], "    ", frequency[i] -
expected)
```

Alternative:

```
def TestRandom (repetitions):
    expected = repetitions / 10      ## initialised as real/integer
                                   ## NextRandom and N defined as integers
    frequency =[0,0,0,0,0,0,0,0,0,0,0] ## defined as an array and
initialised to zero

    for n in range (0,repetitions):
        nextNumber = randint(1, 10)
        frequency[nextNumber] += 1

    print ('The expected frequency is ', expected)

    print ('Number    Frequency    Difference')
    for n in range (1, 11):
        print (n, '    ', frequency[n], '    ', frequency[n] - expected)
```

Alternative:

```
frequency =[0]*11      ## alternate way to initialise array to zero
frequency =[]          ## empty array/list
```

Alternative:

```
for n in range (1,11):
    frequency[n-1] = 0      ##alternate way to initialise array to zero
```

**Alternative:**

```
for n in range (0,11): ##alternate way to initialise array to 2
    frequency.append(0)
```

QUESTION 17.

Cambridge
International
AS & A Level

Cambridge Assessment International Education
Cambridge International Advanced Subsidiary and Advanced Level



COMPUTER SCIENCE

9608/21

Paper 1 Written Paper

October/November 2018

MARK SCHEME

Maximum Mark: 75

Published

This mark scheme is published as an aid to teachers and candidates, to indicate the requirements of the examination. It shows the basis on which Examiners were instructed to award marks. It does not indicate the details of the discussions that took place at an Examiners' meeting before marking began, which would have considered the acceptability of alternative answers.

Mark schemes should be read in conjunction with the question paper and the Principal Examiner Report for Teachers.

Cambridge International will not enter into discussions about these mark schemes.

Cambridge International is publishing the mark schemes for the October/November 2018 series for most Cambridge IGCSE™, Cambridge International A and AS Level components and some Cambridge O Level components.

This document consists of **12** printed pages.



Generic Marking Principles

These general marking principles must be applied by all examiners when marking candidate responses. They should be applied alongside the specific content of the mark scheme or generic level descriptors for a question. Each question paper and mark scheme will also comply with these marking principles.

GENERIC MARKING PRINCIPLE 1:

Marks must be awarded in line with:

- the specific content of the mark scheme or the generic level descriptors for the question
- the specific skills defined in the mark scheme or in the generic level descriptors for the question
- the standard of response required by a candidate as exemplified by the standardisation scripts.

GENERIC MARKING PRINCIPLE 2:

Marks awarded are always **whole marks** (not half marks, or other fractions).

GENERIC MARKING PRINCIPLE 3:

Marks must be awarded **positively**:

- marks are awarded for correct/valid answers, as defined in the mark scheme. However, credit is given for valid answers which go beyond the scope of the syllabus and mark scheme, referring to your Team Leader as appropriate
- marks are awarded when candidates clearly demonstrate what they know and can do
- marks are not deducted for errors
- marks are not deducted for omissions
- answers should only be judged on the quality of spelling, punctuation and grammar when these features are specifically assessed by the question as indicated by the mark scheme. The meaning, however, should be unambiguous.

GENERIC MARKING PRINCIPLE 4:

Rules must be applied consistently e.g. in situations where candidates have not followed instructions or in the application of generic level descriptors.

GENERIC MARKING PRINCIPLE 5:

Marks should be awarded using the full range of marks defined in the mark scheme for the question (however; the use of the full mark range may be limited according to the quality of the candidate responses seen).

GENERIC MARKING PRINCIPLE 6:

Marks awarded are based solely on the requirements as defined in the mark scheme. Marks should not be awarded with grade thresholds or grade descriptors in mind.



Question	Answer			
1(a)(i)	Statement	Selection	Repetition (Iteration)	Assignment
	WHILE Count < 20		✓	
	Count ← Count + 1			✓
	If MyGrade <> 'C' THEN	✓		
	Mark[Count] ← GetMark(StudentID)			✓
	ELSE OUTPUT "Fail"	✓		
	ENDFOR		✓	
	One mark for each row			
1(b)(i)	Statement		Data type	
	MyAverage ← 13.5		REAL	
	ProjectCompleted ← TRUE		BOOLEAN	
	Subject ← "Home Economics"		STRING	
	MyMark ← 270		INTEGER	
	MyGrade ← 'B'		CHAR	
1(b)(ii)	Expression		Evaluates to	
	"Air-" & MID(Subject, 7, 3)		"Air-con"	
	INT(MyAverage / 2)		6	
	ProjectCompleted AND MyMark > 270		FALSE	
	ProjectCompleted OR MyMark > 260		TRUE	
	ASC(MyGrade / 3)		ERROR	

5

5



Question	Answer	
2(a)	<pre> FUNCTION GetDiscountRate(CardNum : STRING) RETURNS REAL DECLARE DRate : REAL DECLARE Points : INTEGER DRate ← 0 Points ← GetPoints(CardNum) IF Points > 199 THEN DRate ← 0.2 ELSE IF Points > 99 THEN DRate ← 0.1 ENDIF ENDIF ENDIF IF Today() = 3 THEN DRate ← DRate * 1.2 ENDIF RETURN DRate ENDFUNCTION </pre> 1 mark for each of the following: 1 Correct FUNCTION heading (as given) and end 2 Declaring local variables for DRate and Points 3 Initialisation of DRate to zero and Points ← GetPoints(CardNum) 4 IF ... THEN ...(ELSE) ... ENDIF with Points > 199 5 (Nested) IF ... THEN ... ENDIF with Points > 99 6 ...correct assignments of DRate to 0.2 and 0.1 7 Checking Today() = 3 and increasing DRate by 20% 8 Return parameter // GetDiscountRate ← DRate Mark points 7 and 8 must not be nested	
2(b)(i)	Name: Syntax Description: Rules of programming language have not been followed Name: Logic Description: Where the program does not behave as expected / does not give the expected result / an error in the logic of the algorithm 1mark for name + 1 mark for corresponding description	2
2(b)(ii)	Name: Stub testing Description: A function could be written for GetPoints() that simply returns a test value or outputs a message (i.e. doesn't do the CardNum lookup)	2



Question	Answer	
2(c)(i)	1 mark for any of the following two values: 0.1 0.2 1.2 99 199 3	
2(c)(ii)	Example: CONSTANT MinDiscount = 0.1 1 mark for each of the following: • meaningful identifier name and corresponding value • correct syntax	2
2(c)(iii)	1 mark for: • The value cannot accidentally get changed // be different in two places • A change to the value requires changing in one place only / don't have to repeatedly write out the same value throughout the program	2
2(c)(iv)	Tried and tested // pre compiled (contains no syntax errors)	1
2(c)(v)	1 mark for feature (Name) and 1 mark for corresponding description (explanation) Example: Name: Meaningful variable names Explanation: To reduce the risk of referring to the wrong variable / make the code easier to understand Name: Indentation Explanation: To see where loops / selection start / end // indicate program structure Name: Variable type-checking as part of module interface Explanation: Reduces the risk of using an incorrect parameter Name: Pretty-Printing Explanation: Highlights the error / auto-complete / type checking Name: / <u>Dynamic</u> Syntax Checking Explanation: Highlights the error as code is typed in	2



Question	Answer																
3(a)	Code has to be in machine code (or equivalent) to be executed																
3(b)	1 mark for the name (what you do) and one for description (how) For example: Method: • Dry run the code // use of white box testing // trace tables • Trace the contents of variables // trace all possible routes through the program Method: • Breakpoints • Run the code to a set point to find error Method: • Variable watch • Check the contents of variables at specific points in the program Method: • Stepping • Execute the code line by line Method: • Include OUTPUT statements in the code • to display the value of variables as the code was running																
3(c)	<table border="1"> <thead> <tr> <th data-bbox="264 1149 986 1211">Statement</th><th data-bbox="986 1149 1166 1211">White-box</th><th data-bbox="1166 1149 1347 1211">Black-box</th></tr> </thead> <tbody> <tr> <td data-bbox="264 1211 986 1312">The student does not need to know the structure of the code.</td><td data-bbox="986 1211 1166 1312"></td><td data-bbox="1166 1211 1347 1312">✓</td></tr> <tr> <td data-bbox="264 1312 986 1413">The student chooses data to test every possible path through the code.</td><td data-bbox="986 1312 1166 1413">✓</td><td data-bbox="1166 1312 1347 1413"></td></tr> <tr> <td data-bbox="264 1413 986 1514">The student chooses normal, boundary and erroneous data.</td><td data-bbox="986 1413 1166 1514">✓</td><td data-bbox="1166 1413 1347 1514">(✓)</td></tr> <tr> <td data-bbox="264 1514 986 1608">The student chooses data to test that the program meets the specification.</td><td data-bbox="986 1514 1166 1608"></td><td data-bbox="1166 1514 1347 1608">✓</td></tr> </tbody> </table> 1 mark per row	Statement	White-box	Black-box	The student does not need to know the structure of the code.		✓	The student chooses data to test every possible path through the code.	✓		The student chooses normal, boundary and erroneous data.	✓	(✓)	The student chooses data to test that the program meets the specification.		✓	4
Statement	White-box	Black-box															
The student does not need to know the structure of the code.		✓															
The student chooses data to test every possible path through the code.	✓																
The student chooses normal, boundary and erroneous data.	✓	(✓)															
The student chooses data to test that the program meets the specification.		✓															



Question	Answer	
4(a)(i)	The identifier name of a global integer referenced	NumElements
	The identifier name of a user-defined procedure	SaveToFile
	The line number of an unnecessary statement	16
	The scope of <code>ArrayString</code>	Local
4(a)(ii)	1 mark for each mark point: - Loop / repeat / iterate through array <code>ResultArray</code> one element at a time - extract a string from <u>row / column 1</u> of the array - compare the string with <code>SearchString</code> - if they match, call <code>SaveToFile()</code> and increment <code>NumberFound</code>	4
4(b)	Pseudocode solution included here for development and clarification of mark scheme. Programming language solutions appear at the end of this mark scheme. <pre> FUNCTION ScanArray(SearchString : STRING) RETURNS INTEGER DECLARE ArrayIndex : INTEGER DECLARE ArrayString : STRING DECLARE NumberFound : INTEGER NumberFound ← 0 FOR ArrayIndex ← 1 TO NumElements ArrayString ← ResultArray[ArrayIndex, 1] IF TO_UPPER(ArrayString) = TO_UPPER(SearchString) THEN CALL SaveToFile(ArrayString) NumberFound ← NumberFound + 1 ENDIF ENDFOR RETURN NumberFound ENDFUNCTION </pre> 1 mark for each of the following: - Function header and end including parameter and return - Declaration of two local variables as above but NOT <code>NumElements</code> - FOR ... ENDFOR loop with range as given - Referencing each element from the array - Converting both strings to uppercase / lowercase - If strings are equal then Call <code>SaveToFile()</code> and increment <code>NumberFound</code>	6



Question	Answer	
4(c)	1 mark for name; 1 mark for each advantage (max 2) Name: Stepwise refinement // Top-down design // Modularisation // Decomposition Advantage: • Makes the problem / task / algorithm easier to understand // reduce program complexity • Smaller modules easier to develop / test / debug • Programmers can work on different modules // different expertise	
4(d)	Pseudocode solution included here for development and clarification of mark scheme. Programming language solutions appear at the end of this mark scheme. <pre> DECLARE ResultArray : ARRAY [1:100, 1:2] OF STRING DECLARE i, j : INTEGER FOR i ← 1 to 100 FOR j ← 1 to 2 ResultArray[i, j] ← '*' ENDFOR ENDFOR </pre> One mark for: • <code>ResultArray</code> declaration / commented in Python • assigning to all elements • assignment of <code>'*'</code>	3



Question	Answer
5	<pre> FUNCTION SaveStatus() RETURNS BOOLEAN DECLARE Time : STRING DECLARE Fuel : STRING DECLARE Distance : STRING DECLARE FileData : STRING DECLARE Tries : INTEGER DECLARE ReturnFlag : BOOLEAN Tries ← 1 ReturnFlag ← TRUE Distance ← GetDistance() Fuel ← GetFuel() Time ← GetTime() WHILE Time = NULL AND Tries < 3 Time ← GetTime() Tries ← Tries + 1 ENDWHILE IF Time = NULL THEN ReturnFlag ← FALSE ELSE FileData ← Time & ',' & Fuel & ',' & Distance OPENFILE "CarStatus.txt" FOR APPEND WRITEFILE "CarStatus.txt", FileData CLOSEFILE "CarStatus.txt" ENDIF RETURN ReturnFlag ENDFUNCTION </pre> 1 mark for each of the following: 1 Function heading as shown 2 Declare Time local variable as STRING 3 Calls GetDistance() and GetFuel() once 4 Loop (up to three times or) until Time <> NULL 5 Call GetTime() in a loop 6 Return FALSE if 3 NULLs 7 Open file in APPEND mode 8 Forming the text string with comma separators and write to the file 9 OPEN ... WRITE ... CLOSE as three lines not separated by loop 10 Return TRUE



Program Code Solutions

Q4 (b): Visual Basic

```
Function ScanArray(SearchString As String) As Integer

    Dim ArrayIndex As Integer
    Dim ArrayString As String
    Dim NumberFound As Integer

    NumberFound = 0

    For ArrayIndex = 1 To NumElements
        ArrayString = ResultArray(ArrayIndex, 1)
        If UCase(ArrayString) = UCase(SearchString) Then
            Call SaveToFile(ArrayString)
            NumberFound = NumberFound + 1
        End If
    Next ArrayIndex
    Return NumberFound

End Function
```

Q4 (b): Pascal

```
function ScanArray(SearchString : String) : Integer;

var
    ArrayIndex : Integer;
    ArrayString : String;
    NumberFound : Integer;

begin
    NumberFound := 0;

    For ArrayIndex := 1 To NumElements do
        begin
            ArrayString := ResultArray[ArrayIndex, 1];
            If ToUpper(ArrayString) = ToUpper(SearchString) then
                begin
                    SaveToFile(ArrayString); // Keyword "Call" not valid
                    NumberFound := NumberFound + 1;
                end;
            end;
        end;

    Result := NumberFound; // ScanArray := NumberFound
end.
```

**Q4 (b): Python**

```
def ScanArray(SearchString):  
  
    # ArrayIndex : integer  
    # ArrayString : string  
    # NumberFound : integer  
  
    NumberFound = 0  
  
    for ArrayIndex in range(NumElements): # 0 to NumElements-1  
        ArrayString = ResultArray[ArrayIndex][0]  
        if ArrayString.upper == SearchString.upper:  
            SaveToFile(ArrayString) # Keyword "Call" not valid  
            NumberFound = NumberFound + 1  
  
    return NumberFound # ScanArray := NumberFound
```

**Q4 (d): Visual Basic**

```
Dim ResultArray(100, 2) As String
Dim i, j As Integer

For i = 1 to 100
    For j = 1 to 2
        ResultArray(i, j) = '*'
    Next j
Next i
```

Q4 (d): Pascal

```
var
    ResultArray : array[1..100, 1..2] of string;
    i, j : integer;
begin
    For i := 1 to 100 do
        For j := 1 to 2 do
            begin
                ResultArray[i, j] := '*';
            end;
        end;
    end.
```

Q4 (d): Python

```
# ResultArray[1..100, 1..2] : String

ResultArray = [[''] * 2 for i in range(100)]

for i in range(100):
    for j in range(2):
        ResultArray[i][j] = ''
```

Q4 (d): Python – alternative 1 of n

```
# ResultArray[1..100, 1..2] : String

ResultArray = [['*'] * 2 for i in range(100)]
```

Q4 (d): Python – alternative 2 of n

```
# ResultArray[1..100, 1..2] : String

ResultArray = [['*'] * 2] * 100
```





Question	Answer	
4(b)	<pre> DECLARE Code : ARRAY[1:500, 1:4] OF STRING DECLARE RowIndex : INTEGER DECLARE ColIndex : INTEGER FOR RowIndex ← 1 TO 500 FOR ColIndex ← 1 TO 4 Code[RowIndex, ColIndex] ← "Empty" ENDFOR ENDFOR </pre> One mark for each of the following: 1 Array declaration 2 Additional local variable 3 Nested loops 4 Array element assignment within the inner loop RowIndex and ColIndex can be interchangeable	
4(c)	Adaptive Maintenance	1

QUESTION 19.



Question	Answer
6(a)	'Pseudocode' solution included here for development and clarification of mark scheme. Programming language example solutions appear in the Appendix. <pre> FUNCTION SearchFile (SearchString : STRING) RETURNS STRING DECLARE FileData : STRING DECLARE Found : BOOLEAN DECLARE SearchLength : INTEGER Found ← FALSE SearchLength ← LENGTH(SearchString) OPENFILE "StudentContact.txt" FOR READ WHILE NOT EOF("StudentContact.txt") AND NOT Found READFILE "StudentContact.txt", FileData IF SearchString = LEFT(FileData, SearchLength) THEN Found ← TRUE ENDIF ENDWHILE CLOSEFILE "StudentContact.txt" IF NOT FOUND THEN RETURN "" ELSE RETURN FileData ENDIF ENDFUNCTION </pre> One mark for each of the following: 1. Function header and end (where appropriate). Parameter optional but if present must be of type STRING 2. Calculate length of string from parameter // extract substring from file line 3. File OPEN() in READ mode and subsequent CLOSE() 4. WHILE loop repeating until EOF() 5. read a line from the file in a loop 6. compare name from file with SearchString in a loop 7. exit loop if SearchString found 8. Return the line from the file if SearchString found or an empty string if not found



Question	Answer	
6(b)	<pre> FUNCTION ProcessArray() RETURNS INTEGER DECLARE NoTelNumber : INTEGER DECLARE Index : INTEGER DECLARE ThisName : STRING DECLARE StudentData : STRING NoTelNumber ← 0 FOR Index ← 1 to 40 ThisName ← ClassList[Index] IF ThisName <> "" //Skip blanks THEN StudentData ← SearchFile(ThisName) IF StudentData = "" //Student not found THEN StudentData ← ThisName & "*No number" NoTelNumber ← NoTelNumber + 1 ENDIF CALL AddToFile(StudentData, "ClassContact.txt") ENDIF ENDIF ENDFOR RETURN NoTelNumber ENDFUNCTION </pre> One mark for each of the following: 1. Function header and end, including return parameter 2. Declaration and initialisation of local count variable (NoTelNumber) 3. FOR loop for 40 array elements ... 4. skip empty elements in a loop 5. use SearchFile(ThisName) and save return value in a loop 6. if Searchfile() returns an empty string, add "*No number" to SearchString ... 7. ... and increment count 8. call AddToFile with both parameters as above in a loop 9. Return count outside the loop	
6(c)	'Pseudocode' solution included here for development and clarification of mark scheme. Programming language example solutions appear in the Appendix. <pre> <u>FUNCTION ProcessArray</u> (<u>ClassList : ARRAY, ClassContact :</u> <u>STRING</u>) <u>RETURNS : INTEGER</u> </pre> One mark per underlined section.	3

**Program Code Example Solutions****Q6 (a): Visual Basic**

```
Function SearchFile(SearchString As String) As String
    Dim FileData As String
    Dim Found As Boolean
    Dim SearchLength As Integer
    Dim FileName As String

    Found = False
    SearchLength = Len(SearchString)

    FileName = "StudentContact.txt"
    FileOpen(1, FileName, OpenMode.Input)
    While Not EOF(1) And NOT Found
        FileData = LineInput(1)
        If SearchString = Left(FileData, SearchLength) Then
            Found = True
        End If
    End While

    FileClose(1)

    If Not Found Then
        Return ""
    Else
        Return FileData
    End If

End Function
```

Alternative:

```
Function SearchFile (SearchString As String) As String
    Dim FileData As String
    Dim Found As Boolean
    Dim SearchLength As Integer
    Dim MyFile As System.IO.StreamReader

    MyFile = My.Computer.FileSystem.OpenTextFileReader("StudentContact.txt")
    Found = False
    SearchLength = Len(SearchString)

    Do While MyFile.Peek <> -1
        FileData = MyFile.Readline()
        If SearchString = LEFT(FileData, SearchLength) Then
            Found = True
            return(FileData)
        End If
    Loop
    MyFile.Close
    If NOT Found then
        return ( "")
    End If
End Function
```

**Q6 (a): Pascal**

```
function SearchFile(var SearchString: string):string;
    var Found : Boolean;
        SearchLength : integer;
        FileData : string;
        MyFile : text;

begin
    Found := False;
    SearchLength := Length(SearchString);

    Assign(MyFile, 'StudentContact.txt');
    Reset(MyFile);

    While NOT EOF(MyFile) AND Found = False do
        Begin
            Readln(MyFile, FileData);
            If SearchString = LeftStr(FileData,SearchLength) then
                Found := True;
            End;
            Close(MyFile);

        If NOT Found then
            SearchFile := ''
        else
            SearchFile := FileData;

    End;
```

Q6 (a): Python

```
def searchFile(searchString):
    ##Declare filedata : string, found : boolean, searchLength : integer
    ##returns a string value

    found = False
    searchLength = len(searchString)
    myFile = open("StudentContact.txt", 'r')
    fileData = myFile.readline()
    while found == False:
        fileData = myFile.readline()
        if not fileData.strip(): #check if no data/end of file
            break
        else:
            if searchString == fileData[0:searchLength]:
                found = True
                print(searchString)

    myFile.close
    if found == False:
        return("")
    else:
        return(fileData)
```

**Q6 (c): Visual Basic**

Function ProcessArray (ClassList() As String, ClassContact As String) As Integer

OR

Function ProcessArray (ClassList As String(), ClassContact As String) As Integer

Q6 (c): Pascal

function ProcessArray (var ClassList:CList; ClassContact:string) :integer;

CList is user-defined type – could be any name that's not a keyword

Q6 (c): Python

def ProcessArray (ClassList, ClassContact) :

QUESTION 20.



Question	Answer
6(a)	To retain data when the computer is shut down / turned off // after the program ends Accept equivalent answer.
6(b)	'Pseudocode' solution included here for development and clarification of mark scheme. Programming language example solutions appear in the Appendix. <pre> FUNCTION SearchFileNtoZ(AccNum : STRING) RETURNS BOOLEAN DECLARE FileData : STRING DECLARE Found : BOOLEAN CONSTANT SearchFile = "UserListNtoZ.txt" Found ← FALSE OPENFILE SearchFile FOR READ WHILE NOT EOF(SearchFile) AND NOT Found READFILE SearchFile, FileData IF AccNum & '*' = LEFT(FileData, LENGTH(AccNum)+ 1) THEN Found ← TRUE ENDIF ENDWHILE CLOSEFILE SearchFile RETURN Found ENDFUNCTION </pre> One mark for each of the following: 1. Function heading and ending, (ignore parameter) and returned BOOLEAN 2. File OPEN UserListNtoZ.txt in READ mode and CLOSE 3. Conditional loop repeating until EOF() or 'Found' 4. Read a line from the file in a loop 5. Compare the correct number of characters with AccNum in a loop 6. Set termination logic if found in a loop 7. Return Boolean value



Question	Answer
6(c)	<pre> PROCEDURE FindDuplicates() DECLARE Index : INTEGER DECLARE FileData : STRING DECLARE Continue : BOOLEAN DECLARE AccNum : STRING Index ← 1 // assuming array is [1:100] Continue ← TRUE OPENFILE "UserListAtoM.txt" FOR READ WHILE NOT EOF("UserListAtoM.txt") AND Continue = TRUE READFILE "UserListAtoM.txt", FileData IF MID(FileData, 7, 1) = '*' // six character reference THEN AccNum ← LEFT(FileData, 6) ELSE AccNum ← LEFT(FileData, 9) ENDIF IF SearchFileNtoZ(AccNum) = TRUE THEN IF Index = 101 // is the array already full? THEN OUTPUT "Error - Array Full" Continue ← FALSE ELSE Duplicates[Index] ← AccNum Index ← Index + 1 ENDIF ENDIF ENDIF ENDWHILE CLOSEFILE "UserListAtoM.txt" ENDPROCEDURE </pre> One mark for each of the following (max 8): 1. Declaration and Initialisation of Index and used to index array Duplicates 2. OPEN file UserListAtoM.txt in READ mode and CLOSE 3. Pre-Condition loop to go through the file until EOF() and early termination if array full 4. Read line from file and extract account number (AccNum) in a loop 5. Call SearchFileNtoZ (with AccNum) following an attempt at MP4 in a loop 6. Check if return value is TRUE and if so: in a loop 7. store AccNum in correct array element 8. increment array index following an attempt at MP7 9. If array overflow OUTPUT error message



Question	Answer	
6(d)(i)	<pre>PROCEDURE ClearArray(BYREF ThisArray : ARRAY, NumElements : INTEGER, InitVal : STRING) DECLARE Index : INTEGER FOR Index ← 1 TO NumElements ThisArray[Index] ← InitVal ENDFOR ENDPROCEDURE</pre> Mark as follows: • Procedure header • Loop • Assignment within loop	
6(d)(ii)	'Pseudocode' solution included here for development and clarification of mark scheme. Programming language example solutions appear in the Appendix. <pre><u>CALL ClearArray(Duplicates, 100, "Empty")</u></pre> Mark as follows: • Procedure call • Parameter list (in brackets)	2

**Program Code Example Solutions****Question 6(b): Visual Basic**

```
Function SearchFileNtoZ(ByVal SearchString As String) As Boolean
    Dim FileData As String
    Dim Found As Boolean

    Found = FALSE

    FileOpen(1, "UserListNtoZ.txt", OpenMode.Input)

    While Not EOF(1) And Not Found

        Filedata = LineInput(1)
        If SearchString & '*' = Left(FileData, Len(SearchString)+1) Then
            Found = TRUE
        End If

    End While

    FileClose(1)

    Return Found

End Function
```

Question 6(b): Pascal

```
function SearchFileNtoZ (SearchString : string): boolean;
var
    FileData : string;
    Found : boolean;
    MyFile : text;

begin
    Found := FALSE;

    assign(MyFile, "UserListNtoZ.txt");
    reset (Myfile);

    while Not EOF(MyFile) And Not Found do
        begin
            readLn(MyFile, FileData);
            if SearchString + '*' = LeftStr(FileData, length(SearchString)+1)
then
                Found := TRUE;

            end;

        close(MyFile);

        result := Found; // SearchFileB := Found;

    end;
```

**Question 6(b): Python**

```
def SearchFileNtoZ(SearchString):  
    ## FileData : String  
    ## Found : Boolean  
    ## MyFile : Text  
  
    Found = False  
  
    MyFile = open("UserListNtoZ.txt", 'r')  
    FileData = MyFile.readline()  
    while FileData != "" and not Found :  
        if SearchString + '*' == FileData[0: len(SearchString)+1]:  
            Found = True  
  
        FileData = MyFile.readline()  
  
    MyFile.close  
  
    return(Found)
```

Question 6(d)(ii): Visual Basic

Call ClearArray(Duplicates, 100, "Empty") 'Call optional

Question 6(d)(ii): Pascal

ClearArray(Duplicates, 100, 'Empty');

Question 6(d)(ii): Python

ClearArray(Duplicates, 100, "Empty")

QUESTION 21.



Question	Answer	Marks
6(a)(i)	One mark per underlined section: <u>DECLARE Result : ARRAY</u> <u>[0:99, 0:1]</u> <u>OF STRING</u>	3



Question	Answer
6(a)(ii)	'Pseudocode' solution included here for development and clarification of mark scheme. Programming language example solutions appear in the Appendix. <pre> FUNCTION FindBooksBy(SearchAuthor : STRING) RETURNS INTEGER DECLARE Title : STRING DECLARE Author : STRING DECLARE Isbn : STRING DECLARE Location : STRING DECLARE Count : INTEGER Count ← 0 OPENFILE "Library.txt" FOR READ WHILE NOT EOF ("Library.txt") READFILE "Library.txt", Title READFILE "Library.txt", ThisAuthor READFILE "Library.txt", ISBN READFILE "Library.txt", Location IF SearchAuthor = ThisAuthor THEN Result[Count, 0] ← Title Result[Count, 1] ← Location Count ← Count + 1 ENDIF ENDWHILE CLOSEFILE("Library.txt") RETURN Count ENDFUNCTION </pre> One mark for each of the following: 1 Function heading (and ending) including parameters 2 Declaration of variables used 3 Open file for reading (Allow <code>Library</code> or <code>Library.txt</code>) 4 <code>WHILE</code> loop checking for <code>EOF()</code>: 5 Read all information 'fields', in the correct order, in a loop 6 If the author matches write <code>Title</code> and <code>Location</code> to <code>Result</code> array in a loop 7 And increment array index in a loop // number found 8 Close file and <code>RETURN Count</code>



Question	Answer
6(b)	<pre> PROCEDURE DisplayResults (Author:STRING, Count:INTEGER) DECLARE Index, GLen : INTEGER DECLARE Gap : STRING Gap ← " " // 25 spaces IF Count = 0 THEN OUTPUT "Search found no books by: " & Author ELSE OUTPUT "Books written by: " & Author OUTPUT "Title" & LEFT(Gap, 20) & "Location" FOR Index ← 1 TO Count GLen ← 25 - LENGTH(Result[Index, 0]) OUTPUT Result[Index, 0] & LEFT(Gap, GLen) & Result[Index, 1] ENDFOR OUTPUT "Number of titles found: " & NUM_TO_STRING(Count) ENDIF ENDPROCEDURE </pre> One mark for each of the following (max 7): 1 Procedure heading and ending including parameters 2 Declaration of local INTEGER variable for use as index 3 Test if count = 0 and if so output suitable message including Author for no books found otherwise output the two header strings (exact format not important) 4 A FOR loop for Count times 5 ... output two array elements from Result array in a loop 6 Final output statement 7 A reasonable attempt at calculating the number of spaces required to align 'Location' column: 8 Alignment correct



Program Code Example Solutions

Q6 (a) (ii): Visual Basic

```
FUNCTION FindBooksBy(ByVal SearchAuthor As String) As Integer

    Dim Title As String
    Dim Author As String
    Dim Isbn As String
    Dim Location As String
    Dim Count As Integer

    Count = 0

    FileOpen(1, "Library.txt", OpenMode.Input)

    While Not EOF(1)
        Title = LineInput(1)
        ThisAuthor = LineInput(1)
        Isbn = LineInput(1)
        Location = LineInput(1)

        If SearchAuthor = ThisAuthor Then
            Result(Count, 0) = Title
            Result(Count, 1) = Location
            Count = Count + 1
        End If
    End While

    FileClose(1)

    Return Count
END FUNCTION
```

**Q6 (a) (ii): Pascal**

```
function FindBooksBy(SearchAuthor : string) : integer;

var
  Title : string;
  Author : string;
  Isbn : string;
  Location : string;
  Count : integer;
  MyFile : text;

begin
  Count := 0;
  assign(MyFile, 'Library.txt');
  reset(MyFile);

  while not EOF(MyFile) do
  begin
    readln(MyFile, Title);
    readln(MyFile, ThisAuthor);
    readln(MyFile, Isbn);
    readln(MyFile, Location);

    if SearchAuthor = ThisAuthor then
    begin
      Result[Count, 0] := Title;
      Result[Count, 1] := Location;
      Count := Count + 1;
    end;

  end;

  close(MyFile);
  FindBooksBy := Count;

end;
```

**Q6 (a) (ii): Python**

```
def FindBooksBy(SearchAuthor):

    ## Title : STRING
    ## Author : STRING
    ## Isbn : STRING
    ## Location : STRING
    ## Count : INTEGER

    Count = 0
    MyFile = open("Library.txt", 'r')
    Title = MyFile.readline()

    while Title != "":
        ThisAuthor = MyFile.readline()
        Isbn= MyFile.readline()
        Location = MyFile.readline()
        if SearchAuthor == ThisAuthor.strip():
            Result[Count][0] = Title
            Result[Count][1] = Location
            Count = Count + 1
        Title = MyFile.readline()

    MyFile.close()
    return(Count)
```