

QUESTION 1.



- 2 (a) (i) Displays the menu (choices)
Repeats the prompt and input ...
...the input is a number between 1 and 4 // Checks number is between 1 and 4

"within range" is not enough

[3]

- (ii) ...the input number is validated

[1]

Page 3	Mark Scheme
	Cambridge International AS/A Level – May/June 2015



(b) (i) 3

(ii) Previous design repeated indefinitely // (new design) limits number of attempts

Penalise "Program terminates/closes"

(c) IF Choice = 1 THEN (CALL) ReadFile (1)
 IF Choice = 2 THEN OUTPUT "Add Customer code" (1)
 IF Choice = 3 THEN OUTPUT "Search Customer code" (1)
 IF Choice = 4 THEN END (1)

alternative answer:

mark as follows:

CASE OF <u>Choice</u> // Select CASE <u>Choice</u>	1 mark
1: (CALL) ReadFile	1 mark (allow CASE = 1)
2: OUTPUT " <u>Add Customer code</u> "	1 mark
3: OUTPUT " <u>Search Customer code</u> "	1 mark
4: END	
ENDCASE	

Output strings must match

[max 3]

(d) Mark as follows:

- Choice / NoOfAttempts declared/commented as integer
Must appear within the 'main' program
Allow: different identifier names
- Constant i assigned a value 3
- There is an 'outer' loop to repeatedly display the menu
- Input 'choice' variable
- Three IF statements (or equivalent) for processing menu choices 1, 2 and 3
Note: they must be correctly formed as 'nested' or 'independent'
- Choice 1 calls procedure ReadFile
- Choice 2 outputs "Add Customer Code"
+ Choice 3 outputs "Search Customer Code"
- Outer loop terminates correctly with 'Choice = 4' //or equivalent
- Procedure DisplayMenu shows the four menu options
- Procedure ReadFile is present ...
and contains a single output message 'Read file code'

[max 8]

QUESTION 2.



- 4 (a) (i) ERROR [1]
- (ii) parityerrorcheck [1]
- (iii) Binary Coded Decimal // Binary▼Coded▼Decimal [2]
- (b) (i)

```
OPENFILE "DISPENSERS" FOR WRITE
REPEAT (1 mark)
  OUTPUT "Enter dispenser code (XXXXX to end)"
  INPUT DispenserCode
  IF DispenserCode <> "XXXXX"
    THEN
      OUTPUT "Enter bank code ..."
      INPUT BankCode
      LineString ← CONCAT(DispenserCode, "▼",BankCode)
      // now write the new line to the file
      WRITEFILE ("DISPENSERS"), LineString
    ENDF
UNTIL DispenserCode = "XXXXX"
CLOSE ("DISPENSERS") // CLOSEFILE
OUTPUT "DISPENSERS file now created"
```

 (1 mark)
(1 mark)
(1 mark)
(1 mark) [6]
- (ii) • Bank code/ Dispenser code is digit characters only
• Bank code is exactly 3 digits // Dispenser code is exactly 5 digits
• Range check on Bank code between 1 and 999
// range check on dispenser code between 1 and 99999
- Note: If no reference made to either Bank code or Dispenser code MAX 1 [max 2]
- (iii) data of the existing 15 dispensers will be lost/overwritten [1]
- (iv) Append // Illustrated with program code statement [1]



(c) Mark as follows:

- Variables declared/commented (at least X2) (1 mark)
- Input of 'ThisBank' with prompt (1 mark)

- File open statement (1 mark)
- File mode is 'Input' (1 mark)
- File close

- Loop (Not a FOR loop) (1 mark)
- Until all records considered

- Isolate LineBankCode (1 mark)
- Isolate LineDispenserCode

- Count initialised (1 mark)
- Count incremented (1 mark)

- Output – List of dispenser codes (1 mark)
- Output – dispenser count (1 mark)

[max 10]

Visual Basic ...

```

Dim DispenserRecord As String
Dim DispenserCode As String : Dim Bank As String
Dim DispenserCount As Integer
Dim ThisBank As String
FileOpen(1, "C:\DISPENSERS.txt", OpenMode.Input)

Console.WriteLine()
Console.Write("Which bank ..(Three digit code)? ")
ThisBank = Console.ReadLine

DispenserCount = 0
Do
    DispenserRecord = LineInput(1)
    DispenserCode = Left(DispenserRecord, 5)
    Bank = Mid(DispenserRecord, 7, 3)

    If Bank = ThisBank Then
        DispenserCount = DispenserCount + 1
        Console.WriteLine(DispenserCode)
    End If
Loop Until EOF(1)
FileClose(1)

Console.WriteLine()
Console.WriteLine("There are " & DispenserCount & " dispensers
for this bank")

```




Python ...

```
# DispenserLine      - String
# DispenserCode      - String
# Bank               - String
# DispenserCount     - Integer
# ThisBank           - String

MyFile = open("c:\DISPENSERS.txt", "r")

ThisBank = input("Which bank ..(Three digit code)? ")

DispenserCount = 0
while 1:
    DispenserLine = MyFile.readline()
    if not DispenserLine:
        break
    DispenserCode = DispenserLine[0:5]
    # slices chars 0,1,2,3,4
    Bank = DispenserLine[6:9] # slices chars 6,7,8

    if Bank == ThisBank:
        DispenserCount = DispenserCount + 1
        print(DispenserCode)

MyFile.close()
print
print("There are " + str(DispenserCount)
      " dispensers for this bank")
```

Pascal ...

```
var DispenserRecord   : String ;
var DispenserCode     : String ;
var Bank              : String ;
var DispenserCount    : Integer ;
var ThisBank          : String ;
var TheFile           : Text ;

begin
assign(TheFile, 'K:\DISPENSERS.txt') ;
reset(TheFile) ;

WriteLn() ;
Write('Which bank ..(Three digit code)? ') ;
ReadLn(ThisBank) ;
C
DispenserCount := 0 ;
repeat
    readln(TheFile, DispenserRecord) ;
    DispenserCode := Copy(DispenserRecord,1, 5) ;
    Bank := copy(DispenserRecord, 7, 3) ;

    If Bank = ThisBank Then
        begin
            DispenserCount := DispenserCount + 1 ;
```

Page 7	Mark Scheme	
	Cambridge International AS/A Level – May/June 2015	

```
        Writeln(DispenserCode)
    end ;

    until EOF(TheFile) ;
    close(TheFile) ;

writeln() ;
writeln('Dispenser count: ', DispenserCount) ;

readln .
```



QUESTION 3.



7	(a)	(i)	165	[1]
		(ii)	"YES" Quotes optional	[1]
		(iii)	9	[1]
		(iv)	83	[1]



(b) (i) Use of correct identifiers only to score

Declaration/Commenting of variables

```
MyMessage As String
EncryptString As String
i As Integer
NextNum As Integer
```

At least two variables correctly documented

[1]

Input of string ...

Correct syntax (for both prompt and assignment) and ...

Uses the MyMessage identifier

[1]

EncryptString set to 'empty string'

Note: Must suggest 'empty' string

[1]

For loop ...

FOR – NEXT keywords // (Python) correct indentation

Correct start/end boundaries

Note: the end boundary must use the language length
function/method //alternative Python syntax

[1]

[1]

Isolate single character

Use of language functions to calculate new number and

Assigned to NextNum

[1]

[1]

Conversion of NextNum to a character and concatenated
to EncryptString

[1]

Correct syntax for output of EncryptString

[1]

[MAX 8]

SAMPLE CODE

PYTHON

```
MyMessage = input("Enter message : ")
EncryptString = ""
for i in range(0, len(MyMessage)) :
    NextNum = ord(MyMessage[i]) + 3
    EncryptString = EncryptString + chr(NextNum)
print(EncryptString)
```

Alternative solution:

```
MyMessage = input("Enter message : ")
EncryptString = ""
for NextChar in MyMessage :
    NextNum = ord(NextChar) + 3
    EncryptString = EncryptString + chr(NextNum)
print(EncryptString)
```



VB

```
Dim MyMessage, EncryptString As String
Dim NextNum, i As Integer
Console.WriteLine("Enter message : ")
MyMessage = Console.ReadLine()
EncryptString = ""
For i = 1 To Len(MyMessage)
    NextNum = Asc(Mid(MyMessage, i, 1)) + 3
    EncryptString = EncryptString + Chr(NextNum)
Next
Console.WriteLine(EncryptString)
```

Alternatives:

```
GetChar(MyMessage, i)
MyMessage.Substring(i, 1)
```

Allow: Use of InputBox and MsgBox

Alternative solution :

```
Dim MyMessage, EncryptString As String
Dim NextNum, i As Integer
Console.WriteLine("Enter message : ")
MyMessage = Console.ReadLine()
EncryptString = ""
For i = 0 To Len(MyMessage) - 1
    NextNum = Asc(MyMessage.Chars(i)) + 3
    EncryptString = EncryptString + Chr(NextNum)
Next
Console.WriteLine(EncryptString)
```

PASCAL

```
var
    MyMessage, EncryptString : string;
    NextNum, i : integer;
begin
    write('Enter message : ');
    readln(MyMessage);
    EncryptString := '';
    for i := 1 to length(MyMessage) do
        begin
            NextNum := ord(MyMessage[i]) + 3;
            EncryptString := EncryptString + chr(NextNum);
        end;
    writeln(EncryptString);
end.
```

- (ii) For each/every character [1]
 A replacement character is 'calculated' from its ASCII value // or by example ... [1]

QUESTION 4.



QUESTION 5.



5 (a) (i) **Explanation:**

- Easier to separate the two strings // to retrieve / search for / edit (text related to CD)
- Obvious where `CDTitle` ends and `CDArtist` begins

Drawback:

- Takes up more / unnecessary space in the file
- If the string is bigger than 40 characters then data will be lost // string length is limited
- The additional spaces will need to be removed before strings can be used

One mark per bullet

(ii) **Problem:** File mode = `WRITE` / file is opened for writing // by explanation **[3]**

Effect: All existing file lines / contents / data will be overwritten / deleted / lost

Solution: `WRITE` should be changed to `APPEND` (allow meaningful example)

Allow first two mark points to be interchanged – read as one paragraph.



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

```
PROCEDURE OutputLocationList()

    DECLARE CDTitle : STRING
    DECLARE CDArtist : STRING
    DECLARE CDSearch : STRING
    DECLARE FileData : STRING
    DECLARE Total : INTEGER
    DECLARE FileData : STRING

    Total ← 0

    OPENFILE "MyMusic" FOR READ

    OUTPUT "Input Location"
    INPUT CDSearch

    WHILE NOT EOF("MyMusic")
        READFILE ("MyMusic", FileData
        IF RIGHT(FileData, 8) = CDSearch
            THEN
                CDTitle ← LEFT(FileData, 40)
                CDArtist ← MID(FileData(41, 40)
                OUPUT (CDTitle & " - " & CDArtist)
                Total ← Total + 1
            ENDIF
        ENDWHILE

    OUTPUT (Total & " CDs found"
    CLOSEFILE("MyMusic")

ENDPROCEDURE
```

One mark for each of the following:

- Procedure heading and ending
- Declaration AND initialisation of variable used as counter (Total above)
- Prompt and input of location to search for CDSearch (or other name)
- Open file for reading (Allow MyMusic or MyMusic.txt)
- Working conditional loop structure including test for EOF

The following points must be present inside a loop

- Read a line from the file (or read complete file in one e.g. as list)
- Isolate 8 chars representing location **and** compare with CDLocation
- Extract strings representing CDTitle and CDArtist
- Output CDTitle and CDArtist (separator optional) if correct location found
- Increment Total if correct location found

The following points must be present after a loop

- Output message including number of CDs found at location (Total)
- Close file

QUESTION 6.



5	(i) 13 / 13.0	[1]
	(ii) 18.6	[1]
	(iii) TRUE	[1]
	(iv) 32	[1]
	(v) 22	[1]

*** End of Mark Scheme – Example program code solutions follow ***



Appendix – Example program code solutions

3(b)(ii): Visual Basic

```
Dim Surname As String
Dim NextChar As Char
Dim NextCodeNumber As Integer
Dim i As Integer
Dim CustomerID As Integer
Dim SLength As Integer

Console.Write("Key in surname ")
Surname = Console.ReadLine
SLength = Len(Surname)
CustomerID = 0
For i = 1 To SLength
    \\ NextChar is a single character from surname
    NextChar = Mid(Surname, i, 1)
    NextCodeNumber = Asc(NextChar)
    CustomerID = CustomerID + NextCodeNumber
Next

Console.WriteLine("Customer ID is " & CustomerID)
```

3(b)(ii): Pascal

```
Var Surname : string;
    SLength, i, CustomerID, NextCodeNumber : integer;
    NextChar : char;
begin
    Writeln ('Enter the surname: ');
    Readln (Surname);
    SLength := Length(Surname);
    CustomerID := 0;
    For i := 1 to SLength do
        begin
            NextChar := SurName[i];
            NextCodeNumber := Ord(NextChar);
            CustomerID := CustomerID + NextCodeNumber;
        end;
    Writeln ('Customer ID is ', CustomerID);
    Readln;
end.
```



3(b)(ii): Python

```
# Surname String
# NextChar Char
# NextCodeNumber, I, CustomerID, SLength Integer

Surname = input("Key in Surname ")
SLength = len(Surname)
CustomerID = 0

for i in range(SLength):
    # NextChar is a single character from surname
    NextChar = Surname[i]
    NextCodeNumber = ord(NextChar)
    CustomerID = CustomerID + NextCodeNumber

print("Customer ID is " + str(CustomerID))
```

4(e): Visual Basic

```
Function ProductCodeSearch(ByVal SearchCode As String) As Integer
    Dim FoundCode As Integer
    Dim i As Integer

    i = 1
    FoundCode = -1

    Do
        If SearchCode = PCode(i) Then
            FoundCode = i
        Else
            i = i + 1
        End If
    Loop Until i = 1001 Or FoundCode <> -1
    Return FoundCode
End Function
```



4(e): Pascal

```
Function ProductCodeSearch (SearchCode : String): integer;

    var FoundCode, ThisIndex : integer;
        Found : Boolean;

Begin
    Found := false;
    ThisIndex := 1;
    Repeat
        If SearchCode = PCode[ThisIndex] then
            Begin
                FoundCode := ThisIndex;
                Found := true;
            End
            ThisIndex := ThisIndex + 1;
        end;
    Until (ThisIndex = 1001) OR (Found);
    If Found = false then
        FoundCode := -1
    ProductCodeSearch := FoundCode;
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"
            FoundIndex = i
        else:
            i = i + 1

    if Found == "no":
        FoundIndex = -1

    return FoundIndex
```

QUESTION 7.

Cambridge International AS/A Level – October/November 2016



6 (i) 10 / 10.0

(ii) 18.4

(iii) 41

(iv) TRUE

[1]

(v) 12.4

[1]



Appendix – Program code example solutions

Q4 (b): Visual Basic

```
Randomize()
Dim i As Integer
Dim NextNumber As Integer
For i = 1 To 4
    NextNumber = 1 + Int(Rnd() * 150)
    Console.WriteLine(NextNumber)
Next
```

OR

```
Randomize()
Dim Num1, Num2, Num3, Num4 As Integer
Num1 = 1 + Int(Rnd() * 150)
Num2 = 1 + Int(Rnd() * 150)
Num3 = 1 + Int(Rnd() * 150)
Num4 = 1 + Int(Rnd() * 150)
Console.WriteLine(Num1, Num2, Num3, Num4)
```

Q4 (b): Pascal

```
Var i : Integer;
    NextNumber : Integer;
Begin
    Randomize;
    For i := 1 To 4 Do
        Begin
            NextNumber := 1 + Random(150);
            Writeln(NextNumber);
        End;
        Readln;
    End.
```

OR

```
Var Num1, Num2, Num3, Num4 : Integer;
Begin
    Randomize;
    Num1 := 1 + Random(150);
    Num2 := 1 + Random(150);
    Num3 := 1 + Random(150);
    Num4 := 1 + Random(150);
    Writeln(Num1, Num2, Num3, Num4);
    Readln;
End.
```



Q4 (b): Python

```
import random
# i : Integer
# NextNumber : Integer

for i in range(1, 5) :
    NextNumber = 1 + int(150 * random.random())
    print(NextNumber)
```

Alternative:

```
import random
# i Integer
# NextNumber Integer
for i in range(1, 5) :
    NextNumber = random.randint(1, 150)
    print(NextNumber)
```

OR

```
import random
# i Integer
# Num1, Num2, Num3, Num4 Integer

Num1 = random.randint(1, 150)
Num2 = random.randint(1, 150)
Num3 = random.randint(1, 150)
Num4 = random.randint(1, 150)

print(Num1, Num2, Num3, Num4)
```

**Q5 (b) (ii): Visual Basic**

```
Dim PlayerName(8) As String
Dim i As Integer
FileOpen(1, "Names.txt", OpenMode.Input)
i = 1
Do
    PlayerName(i) = LineInput(1)
    i = i + 1
Loop Until EOF(1)
FileClose(1)
```

Alternative:

```
Dim PlayerName(8) As String
Dim i As Integer
FileOpen(1, "Names.txt", OpenMode.Input)
For i = 1 To 8
    PlayerName(i) = LineInput(1)
Next
FileClose(1)
```

Alternative:

```
Dim sr As StreamReader = New StreamReader("Names.txt")
Dim line As String
line = sr.ReadLine()
i = 1
Do While (line <> Nothing)
    PlayerName(i) = line
    i = i + 1
    line = sr.ReadLine()
Loop
sr.Close()
```


Page 13	Mark Scheme
	Cambridge International AS/A Level – October/November 2016



Q5 (b) (ii): Pascal

```

Var Names : TextFile;
    i : Integer;
    PlayerName : Array[1..8] Of String;
Begin
    AssignFile(Names, 'Names.txt');
    Reset(Names);
    i := 1;
    While Not Eof(Names) Do
        Begin
            Readln(Names, PlayerName[i]);
            Writeln(PlayerName[i]);
            i := i + 1;
        End;
    Close(Names);
    Readln;
End.

```

Alternative:

```

Var Names : TextFile;
    i : Integer;
    PlayerName : Array[1..8] Of String;
Begin
    AssignFile(Names, 'Names.txt');
    Reset(Names);
    For i := 1 To 8 Do
        Begin
            Readln(Names, PlayerName[i]);
            Writeln(PlayerName[i]);
        End;
    Close(Names);
    Readln;
End.

```

Page 14	Mark Scheme
	Cambridge International AS/A Level – October/November 2016



Q5 (b) (ii): Python

```
# PlayerName : List
# NextPlayer : String
# File : File handle
File = open("Names.txt", "r")
PlayerName = []
while (1) :
    NextPlayer = File.readline()
    if not NextPlayer :
        break
    else :
        PlayerName.append(NextPlayer)
File.close()
```

Alternative:

```
# PlayerName : List
# NextPlayer : String
# File : File handle
# i : Integer
File = open("Names.txt", "r")
PlayerName = []
for i in range(1, 9) :
    NextPlayer = File.readline()
    PlayerName.append(NextPlayer)
File.close()
```

Alternative:

```
# PlayerName : List
# NextPlayer : String
# File : File handle
# i : Integer
File = open("Names.txt", "r")
PlayerName = [" " for i in range(8)]
for i in range(1, 9) :
    PlayerName[i - 1] = File.readline()
File.close()
```



Q5 (b) (iii): Visual Basic

```

Found = False
i = 1

Do
    If ThisPlayerName = PlayerName(i) Then
        Found = True
        PlayerNumber = i
    Else
        i = i + 1
    End If
Loop Until Found = True Or i = 9

```

Q5 (b) (iii): Pascal

```

Begin
Found := False;
i := 1;
Repeat
    If (ThisPlayerName = PlayerName[i]) Then
        Begin
            Found := True;
            PlayerNumber := i;
        End
    Else
        i := i + 1;
    Until (Found) Or (i = 9);
End.

```

Q5 (b) (iii): Python

```

Found = False
PlayerName = [j.strip() for j in PlayerName]
if ThisPlayerName in PlayerName :
    PlayerNumber = PlayerName.index(ThisPlayerName) + 1
    Found = True

```

Alternative:

```

Found = False
i = 1
while not Found and i < 9 :
    if ThisPlayerName == PlayerName[i].strip() :
        Found = True
        PlayerNumber = i
    else :
        i = i + 1

```

Alternative:

```

Found = False
for i in range(1, 9) :
    if ThisPlayerName == PlayerName[i].strip() :
        Found = True
        PlayerNumber = i

```

QUESTION 8.



Question	Answer
6(a)	Pseudocode solution included here for development and clarification of mark scheme. Programming language solutions appear in the Appendix. <pre> FUNCTION ValidatePassword(InString : STRING) RETURNS BOOLEAN DECLARE LCaseChar, UCaseChar, NumChar, n : INTEGER DECLARE NextChar : CHAR DECLARE ReturnFlag : BOOLEAN ReturnFlag ← TRUE LCaseChar ← 0, UCaseChar ← 0, NumChar ← 0 FOR n ← 1 TO LENGTH(InString) NextChar ← MID(InString,n,1) IF NextChar >= 'a' AND NextChar <= 'z' THEN LCaseChar ← LCaseChar + 1 ELSE IF NextChar >= 'A' AND NextChar <= 'Z' THEN UCaseChar ← UCaseChar + 1 ELSE IF NextChar >= '0' AND NextChar <= '9' THEN NumChar ← NumChar + 1 ELSE ReturnFlag ← False //invalid character ENDIF ENDIF ENDIF ENDIF ENDFOR IF Not (LCaseChar>=2 AND UCaseChar>= 2 AND NumChar>= 3) THEN ReturnFlag ← FALSE ENDIF RETURN (ReturnFlag) ENDFUNCTION </pre> 1 mark for each of the following: 1. Correct Function heading and ending 2. Declaring three counter variables (upper, lower, numeric) 3. Initialising counters 4. Correct loop 5. Picking up NextChar from InString 6. Check and count number of lower case 7. Check and count number of upper case



Question	Answer	
6(a)	8. Check and count number of numeric 9. Check for invalid character 10. Combine all four tests into a single Boolean value 11. Returning correct Boolean value	
6(b)(i)	String1: (e.g. "AAAbb123") One mark for a valid string having: • at least 2 uppercase alphabetic • at least 2 lowercase alphabetic • at least 3 numeric characters • No other character String2 – String5: One mark for correct string and explanation (testing different rules of the function) Test strings breaking different rules: • With incorrect numbers of: • Lower case characters • Upper case characters • Numeric characters • Containing an invalid character	5
6(b)(ii)	White Box	1
6(b)(iii)	• Testing may be carried out before the modules are developed // not ready for full testing • Module stubs contain simple code to provide a known response // temporary replacement for a called module	2

**Programming Solutions****Programming Code Example Solutions****Q5 : Visual Basic**

```
Sub SearchFile()  
    Dim FileData As String  
    Dim SearchID As String  
    Dim ArrayIndex As Integer  
  
    ArrayIndex = 1  
    FileOpen(1, "LoginFile.txt", OpenMode.Input)  
    SearchID = Console.ReadLine()  
  
    Do While Not EOF(1)  
        FileData = LineInput(1)  
        If SearchID = LEFT(FileData, 5) Then  
            LoginEvents(ArrayIndex, 1) = Mid(Filedata, 6, 4)  
            LoginEvents(ArrayIndex, 2) = Right(Filedata, 14)  
            ArrayIndex = ArrayIndex + 1  
        End If  
    Loop  
    FileClose(1)  
End Sub
```

Alternative:

```
Sub SearchFile()  
    Dim FileData As String  
    Dim SearchID As String  
    Dim ArrayIndex As Integer  
    Dim MyFile As System.IO.StreamReader  
  
    ArrayIndex = 1  
    MyFile = Mycomputer.FileSystem.OpenTextFileReader("Loginfile.txt")  
    SearchID = Console.ReadLine()  
  
    Do While MyFile.Peek < > -1  
        FileData = MyFile.ReadLine()  
        If SearchID = LEFT(FileData, 5) Then  
            LoginEvents(ArrayIndex, 1) = Mid(Filedata, 6, 4)  
            LoginEvents(ArrayIndex, 2) = Right(Filedata, 14)  
            ArrayIndex = ArrayIndex + 1  
        End If  
    Loop  
    MyFile.Close  
End Sub
```

**Q5 : Pascal**

```
Procedure SearchFile();
```

```
var FileData      : String;  
var SearchID      : String;  
var ArrayRow      : Integer;  
Var MyFile        : Text;
```

```
Begin
```

```
    ArrayRow := 1;  
    Assign(MyFile, "Loginfile.txt");  
    Reset(MyFile);  
    Readln(SearchID);
```

```
    While NOT EOF(MyFile) do
```

```
    Begin
```

```
        Readln(MyFile, FileData)  
        IF SearchID = LeftStr(FileData,5) then  
            Begin  
                LoginEvents[ArrayRow,1] = Copy(FileData,6,4);  
                LoginEvents[ArrayRow,2] = Rightstr(FileData,14);  
                ArrayRow = ArrayRow + 1
```

```
            End;
```

```
    End;
```

```
        Close(MyFile);
```

```
End.
```

**Q5 : Python**

```
def SearchFile():

    # FileData : STRING
    # ArrayRow : INTEGER
    # SearchID : STRING

    ArrayRow = 0
    MyFile = open("Loginfile.txt", 'r')
    SearchID = input()
    FileData = MyFile.readline()

    While FileData != ""
        If SearchID == FileData[:5]           #First 5 characters
            LoginEvents[ArrayRow][1] = FileData[5:9]   #next 4 characters
            LoginEvents[ArrayRow][2] = FileData[-14:]  #last 14 characters
            ArrayRow = ArrayRow + 1
            FileData = MyFile.readline()

    myFile.close()
    return()
```

Alternative:

```
def SearchFile():

    # FileData : STRING
    # ArrayRow : INTEGER
    # SearchID : STRING

    ArrayRow = 0
    Myfile = open("Loginfile.txt", 'r')
    SearchID = input()
    For FileData in MyFile
        IF SearchID == FileData[:5]           #First 5 characters
            LoginEvents[ArrayRow][1] = FileData[5:9]   #next 4 characters
            LoginEvents[ArrayRow][2] = FileData[-14:]  #last 14 characters
            ArrayRow = ArrayRow + 1

    MyFile.close()
    return()
```


**Q6 (a): Visual Basic**

```
Function ValidatePassword(InString As String) As Boolean
    Dim LCaseChar, UCaseChar, NumChar As Integer
    Dim NextChar As Char
    Dim ReturnFlag As Boolean
    Dim n As Integer
    ReturnFlag = TRUE
    LCaseChar = 0
    UCaseChar = 0
    NumChar = 0

    For n = 1 to Len(InString)
        NextChar = Mid(InString, n, 1)
        If NextChar >= 'a' And NextChar <= 'z' Then
            LCaseChar = LCaseChar + 1
        Else
            If NextChar >= 'A' And NextChar <= 'Z' Then
                UCaseChar = UCaseChar + 1
            Else
                If NextChar >= '0' And NextChar <= '9' Then
                    NumChar = NumChar + 1
                Else
                    ReturnFlag = False    //invalid character
                End If
            End If
        End If
    Next

    If NOT (LCaseChar >= 2 And UCaseChar >= 2 And NumChar >= 3) Then
        ReturnFlag = FALSE
    End If

    Return(ReturnFlag)

End Function
```

**Q6 (a): Pascal**

```
Function ValidatePassword(InString : String): Boolean;
  Var LCaseChar, UCaseChar, NumChar : Integer;
  Var NextChar : Char;
  Var ReturnFlag : Boolean;
  Var n : Integer;

  begin
    ReturnFlag := TRUE;
    LCaseChar := 0;
    UCaseChar := 0;
    NumChar := 0;

    For n := 1 to Length(InString) do
      begin
        NextChar := Copy(InString,n,1);
        If NextChar >= 'a' And NextChar <= 'z' Then
          LCaseChar := LCaseChar + 1
        Else If NextChar >= 'A' AND NextChar <= 'Z' Then
          UCaseChar := UCaseChar + 1
        Else If NextChar >= '0' AND NextChar <= '9' Then
          NumChar := NumChar + 1
        Else
          ReturnFlag := False      //invalid character
      end
    end

    If NOT(LCaseChar >= 2 And UCaseChar >= 2 And NumChar >=3) then
      ReturnFlag := False;

  ValidatePassword := ReturnFlag
end;
```

**Q6 (a): Python**

```
def ValidatePassword(InString):
    # lCaseChar, uCaseChar, numChar : INTEGER
    # nextChar : CHAR
    # returnFlag : BOOLEAN
    # n : INTEGER
    returnFlag = TRUE
    lCaseChar = 0
    uCaseChar = 0
    numChar = 0

    for n in range (0, Len(InString))
        nextChar = InString[n]
        If nextChar >= 'a' and nextChar <= 'z':
            lCaseChar = lCaseChar + 1
        ELSE:
            IF nextChar >= 'A' and nextChar <= 'Z':
                uCaseChar = uCaseChar + 1
            ELSE:
                IF nextChar >= '0' and nextChar <= '9':
                    numChar = numChar + 1
                ELSE:
                    returnFlag = False    //invalid character

    IF Not (lCaseChar >= 2 and uCaseChar >= 2 and numChar >= 3):
        returnFlag = FALSE

    Return (returnFlag)

#next code block
```

QUESTION 9.

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>				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 mark per answer																														
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 10.



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.





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 an algorithm 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 algorithm 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 to the next decision. The second decision diamond checks if Number equals EndNumber. If YES, the final count (NumberFound) is output, and the process stops. If NO, the Number is incremented, and the loop repeats from the modulus calculation 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 11.



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><thead><tr><th>Data value</th><th>Data type</th></tr></thead><tbody><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></tbody></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><thead><tr><th>Data</th></tr></thead><tbody><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></tbody></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 12.

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	



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" data-bbox="264 1149 1350 1608"> <thead> <tr> <th>Statement</th><th>White-box</th><th>Black-box</th></tr> </thead> <tbody> <tr> <td>The student does not need to know the structure of the code.</td><td></td><td>✓</td></tr> <tr> <td>The student chooses data to test every possible path through the code.</td><td>✓</td><td></td></tr> <tr> <td>The student chooses normal, boundary and erroneous data.</td><td>✓</td><td>(✓)</td></tr> <tr> <td>The student chooses data to test that the program meets the specification.</td><td></td><td>✓</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 13.

Cambridge
International
AS & A Level

Cambridge Assessment International Education
Cambridge International Advanced Subsidiary and Advanced Level



COMPUTER SCIENCE

9608/22

Paper 2 Written Paper

May/June 2019

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 May/June 2019 series for most Cambridge IGCSE™, Cambridge International A and AS Level and Cambridge Pre-U components, and some Cambridge O Level components.

This document consists of **15** printed pages.



Cambridge Assessment
International Education



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)	- a sequence of steps / stages / instructions - to implement a task // solution to a problem Allow alternatives to sequence providing meaning is clear.														
1(a)(ii)	Input: - e.g. <code>INPUT MyVar // READFILE MyFile, MyString</code> Process: - e.g. <code>NextChar ← 'X' // Count ← Count + 1</code> Output:: - e.g. <code>OUTPUT "Hello World" // WRITEFILE "LogFile.txt", SomeData</code> Mark as follows: One mark for each stage (Input and Process) One mark for each pseudocode example														
1(b)(i)	<table><thead><tr><th>Expression</th><th>Evaluates to</th></tr></thead><tbody><tr><td><code>STRING_TO_NUM(RIGHT(ID, 3))</code></td><td>234.0 / 234</td></tr><tr><td><code>INT(Height * Children)</code></td><td>11</td></tr><tr><td><code>IsMarried AND Married < 31/12/1999</code></td><td>TRUE</td></tr><tr><td><code>LENGTH(ID & NUM_TO_STRING(Height))</code></td><td>8</td></tr><tr><td><code>MID((ID, INT(Height) - Children, 2)</code></td><td>"23"</td></tr></tbody></table> No quotes for row 1 Quotes (single or double) for row 5	Expression	Evaluates to	<code>STRING_TO_NUM(RIGHT(ID, 3))</code>	234.0 / 234	<code>INT(Height * Children)</code>	11	<code>IsMarried AND Married < 31/12/1999</code>	TRUE	<code>LENGTH(ID & NUM_TO_STRING(Height))</code>	8	<code>MID((ID, INT(Height) - Children, 2)</code>	"23"		5
Expression	Evaluates to														
<code>STRING_TO_NUM(RIGHT(ID, 3))</code>	234.0 / 234														
<code>INT(Height * Children)</code>	11														
<code>IsMarried AND Married < 31/12/1999</code>	TRUE														
<code>LENGTH(ID & NUM_TO_STRING(Height))</code>	8														
<code>MID((ID, INT(Height) - Children, 2)</code>	"23"														
1(b)(ii)	<table><thead><tr><th>Variable</th><th>Data type</th></tr></thead><tbody><tr><td><code>Married</code></td><td>DATE</td></tr><tr><td><code>ID</code></td><td>STRING</td></tr><tr><td><code>MiddleInitial</code></td><td>CHAR</td></tr><tr><td><code>Height</code></td><td>REAL</td></tr><tr><td><code>IsMarried</code></td><td>BOOLEAN</td></tr></tbody></table> One mark per data type	Variable	Data type	<code>Married</code>	DATE	<code>ID</code>	STRING	<code>MiddleInitial</code>	CHAR	<code>Height</code>	REAL	<code>IsMarried</code>	BOOLEAN		5
Variable	Data type														
<code>Married</code>	DATE														
<code>ID</code>	STRING														
<code>MiddleInitial</code>	CHAR														
<code>Height</code>	REAL														
<code>IsMarried</code>	BOOLEAN														



Question	Answer	
2(a)(i)	- To make a more manageable / understandable solution - To support modular design	
2(a)(ii)	- Allows the subroutine to be called from many / multiple places - Subroutine may be (independently) tested and debugged - If the task changes the change needs to be made only once - Reduces unnecessary duplication / program lines - Allows teams to work on different parts of the solution	
2(a)(iii)	Type of subroutine: Function Justification: It returns a value // assigns a value to variable <i>Answer</i> One mark for type One mark for justification	2
2(b)	- An editor is used to produce / write / modify the <u>source code</u> / <u>program</u> / <u>high-level language code</u> OR by example: An editor provides (features such as) context-sensitive prompts / dynamic syntax checking etc. - A translator (compiler) is used to translate / convert the source code / program / high-level language code into <u>object code</u> / <u>machine code</u> / <u>an executable file</u>. OR A translator (interpreter) is used to translate the source code / program / high-level language code <u>line by line</u> - A debugger is used to test the program / detect errors (and correct errors) in the program. One mark per bullet point	3



Question	Answer
2(c)	Control structure: A (pre-) <u>conditional</u> loop Function of code: • Check if <code>Result</code> is less than 20 and If true, calls <code>ResetSensor</code> with parameter value 3... • ... and assign the value returned by <code>GetSensor</code> with parameter value 3 to <code>Result</code> • Loop until <code>Result >= 20</code> OR Control structure: A selection // conditional statement Function of code: • Check if <code>Result</code> is less than 20 and If true, calls <code>ResetSensor</code> with parameter value 3... • ... and assign the value returned by <code>GetSensor</code> with parameter value 3 to <code>Result</code> One mark for control structure, maximum two for function Function of code marks independent of answer to control structure

Question	Answer	Marks
3(a)(i)	<u>PROCEDURE SubA</u> (<u>A : STRING</u>, <u>B : INTEGER</u>, <u>BYREF C : CHAR</u>) One mark for each underlined part Ignore <code>BYVAL</code> for parameter A and/or parameter B Parameter order / names not important but must be correct data types	3
3(a)(ii)	<u>Function SubB</u> (<u>D : STRING</u>, <u>E : INTEGER</u>) <u>RETURNS BOOLEAN</u> One mark for each underlined part Ignore <code>BYVAL</code> for parameter D and/or parameter E Parameter order / names not important but must be correct data types	3
3(b)	• Selection • Iteration • Sequence One mark per bullet to max. 2	2



Question	Answer				
4(a)(i)	Index	NextChar	Selected	NewValue	NewString
			0		"0"
	1	'1'			"01"
	2	'2'			"012"
	3	'∇'		12	
			12		
					"0"
	4	'3'			"03"
	5	'4'			"034"
	6	'∇'		34	
			34		
					"0"
	7	'5'			"05"
	8	'∇'		5	
					"0"
	9	'∇'		0	
					"0"
	10	'3'			"03"
	11	'9'			"039"
One mark for each column. If no mark for columns, award one mark for initialisation of <i>Selected</i> to 0 and <i>Newstring</i> to '0' (single or double quotes).					
4(a)(ii)	34				
					1



Question	Answer	
4(b)(i)	- The final value (in the string) is the largest value (39) and is not considered // final comparison with variable <code>Selected</code> is not made - The loop terminates at the end of the string (the character 9) // there wasn't a final space / non-numeric digit One mark per bullet.	
4(b)(ii)	- Check the (final) value of <code>NewString</code> after the loop... ...and see if it is greater than <code>Selected</code> (repeat the existing conditional clause) OR - Amend the algorithm to add a space character / non-numeric character to the end of the string... ...before the <code>FOR</code> loop / at the start of the function One mark per bullet point Accept alternative workable solution	2



Question	Answer	
5(a)	One mark for each of: • Open the file • Set a count to zero • Loop until end of file // no more lines to read • Increment the count each time a line is read in a loop Maximum 3 marks	
5(b)	<pre>PROCEDURE CountLines(FileName : STRING) DECLARE NumLines : INTEGER DECLARE Dummy : STRING NumLines ← 0 OPENFILE FileName FOR READ WHILE NOT EOF(FileName) READFILE FileName, Dummy NumLines ← NumLines + 1 ENDWHILE CLOSEFILE FileName OUTPUT "Number of lines in the file : ", NumLines ENDPROCEDURE</pre> One mark for each of the following: 1. Procedure header and end, including parameter 2. Declaration and initialisation of a local INTEGER to count lines (e.g. NumLines) 3. OPEN file in read mode and CLOSE file 4. WHILE loop stopping when EOF(FileName) 5. Read a line from the file and increment NumLines in a loop 6. Output a message plus the NumLines outside a loop	6



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 GetInfo() RETURNS STRING DECLARE ID : STRING DECLARE PreferredName : STRING DECLARE Valid : BOOLEAN Valid ← FALSE WHILE Valid = FALSE OUTPUT "Please Enter a valid ID" INPUT ID IF LENGTH(ID) = 5 AND LEFT(ID, 1) >= 'A' <u> </u> AND LEFT(ID, 1) <= 'Z' <u> </u> AND ISNUM(RIGHT(ID, 4)) <u> </u> THEN Valid ← TRUE ENDIF ENDWHILE OUTPUT "Please enter preferred name" INPUT PreferredName RETURN ID & '*' & PreferredName ENDFUNCTION </pre> One mark for each of the following: 1. Function header and end (where appropriate) 2. Local variables used are declared (commented in python) 3. Prompt and input for ID (until valid) and preferred name 4. Conditional loop repeating while ID is invalid 5. test length in a loop 6. test first character in a loop 7. test last four characters in a loop 8. Concatenate using correct separator character and return resulting string



Question	Answer	
6(b)	'Pseudocode' solution included here for development and clarification of mark scheme. Programming language example solutions appear in the Appendix. <pre> PROCEDURE TopLevel() DECLARE Response : CHAR DECLARE InputData : STRING DECLARE Success : BOOLEAN Response ← 'Y' WHILE Response = 'Y' InputData ← GetInfo() IF LEFT(InputData,1) < 'N' THEN Success ← WriteInfo(InputData, "File1.txt") ELSE Success ← WriteInfo(InputData, "File2.txt") ENDIF IF NOT Success THEN Response ← 'N' ELSE OUTPUT "Enter details for another student? (Y/N)" INPUT Response ENDIF ENDWHILE ENDPROCEDURE </pre> One mark for each of the following: 1. Procedure header and end 2. Conditional loop terminated with user input 3. call to <code>GetInfo()</code> in a loop 4. check first character of returned <code>UserID</code> value in a loop 5. call(s) to <code>WriteInfo()</code> in both cases ... 6. ... with two <code>STRING</code> parameters in a loop 7. exit procedure if <code>WriteInfo()</code> unsuccessful in a loop 8. if <code>WriteInfo()</code> successful, prompt and check input to repeat / exit in a loop	
6(c)	<u>FUNCTION WriteInfo (FileData : STRING, Filename : STRING)</u> <u>RETURNS BOOLEAN</u> One mark per underlined section	3

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

```
Function GetInfo() As String
    Dim ID As String = ""
    Dim PreferredName As String = ""
    Dim Valid As Boolean = False
    While Valid = False
        Console.WriteLine("Please enter a valid ID : ")
        ID = Console.ReadLine()
        If Len(ID) = 5 And Left(ID, 1) >= "A" And Left(ID, 1) <= "Z" ____
            And IsNumeric(Right(ID, 4)) Then
            Valid = True
        End If
    End While
    Console.WriteLine("Please enter preferred name : ")
    PreferredName = Console.ReadLine()
    Return ID & "*" & PreferredName
End Function
```

Alternative:

```
Function GetInfo() As String

    Dim ID As String
    Dim PreferredName As String
    Dim Valid As Boolean
    Dim Number As String
    Dim Size As Integer
    Dim i As Integer

    Valid = False

    While Valid = False
        Console.WriteLine("Please Enter a valid ID")
        ID = Console.ReadLine()
        Size = Len(ID)

        If (Size = 5) And ((Left(ID, 1) >= "A") And (Left(ID, 1) <= "Z"))
Then
            Valid = True
            For i = 2 To 5
                Number = Mid(ID, i, 1)
                If (Number < "0") Or (Number > "9") Then
                    Valid = False
                End If
            Next
        End If
    End While

    Console.WriteLine("Please enter preferred name")
    PreferredName = Console.ReadLine()
    Return (ID & "*" & PreferredName)
End Function
```

**Q6 (a): Pascal**

```
function GetInfo() : String;
var
    ID : String;
    PreferredName : String;
    Valid : Boolean;
    Value, Code : Integer;
begin
    Valid := false;
    while not Valid do
    begin
        Write('Please enter a valid ID : ');
        Readln(ID);
        if (Length(ID) = 5) and (ID[1] >= 'A') and (ID[1] <= 'Z') then
            Valid := true;
        Val(Copy(ID, 2, 4), Value, Code);
        if Code <> 0 then
            Valid := false;
        end;
        Write('Please enter preferred name : ');
        Readln(PreferredName);
        GetInfo := ID + '*' + PreferredName;
    end;
end;
```

Free Pascal

```
function GetInfo() : String;
var
    ID : String;
    PreferredName : String;
    Valid : Boolean;
    Value, Code : Integer;
begin
    Valid := false;
    while not Valid do
    begin
        Write('Please enter a valid ID : ');
        Readln(ID);
        if (Length(ID) = 5) and (ID[1] >= 'A') and (ID[1] <= 'Z') and (IsNumber(SubStr(ID, 2, 4))) then
            Valid := true;
        end;
        Write('Please enter preferred name : ');
        Readln(PreferredName);
        result := ID + '*' + PreferredName;
    end;
end;
```

**Q6 (a): Python**

```
def GetInfo() :  
    ID = ""                # string variable  
    PreferredName = ""    # string variable  
    Valid = False         # Boolean variable  
    while not Valid :  
        ID = input("Please enter a valid ID : ")  
        if len(ID) == 5 and ID[0] >= "A" and ID[0] <= "Z" and  
ID[1:].isnumeric() :  
            Valid = True  
    PreferredName = input("Please enter preferred name : ")  
    return ID + "*" + PreferredName
```

**Q6 (b): Visual Basic**

```
Sub TopLevel()  
    Dim Response As String = "Y"  
    Dim InputData As String = ""  
    Dim Success As Boolean = True  
    While Response = "Y"  
        InputData = GetInfo()  
        If Left(InputData, 1) < "N" Then  
            Success = WriteInfo(InputData, "File1.txt")  
        Else  
            Success = WriteInfo(InputData, "file2.txt")  
        End If  
        If Not Success Then  
            Response = "N"  
        Else  
            Console.Write("Enter details for another student? Y/N ")  
            Response = Console.ReadLine()  
        End If  
    End While  
End Sub
```

Q6 (b): Pascal

```
procedure TopLevel();  
var  
    Response : Char;  
    InputData : String;  
    Success : Boolean;  
begin  
    Response := 'Y';  
    while Response = 'Y' do  
        begin  
            InputData := GetInfo();  
            if InputData[1] < 'N' then  
                Success := WriteInfo(InputData, 'File1.txt')  
            else  
                Success := WriteInfo(InputData, 'File2.txt');  
            if not Success then  
                Response := 'N'  
            else  
                begin  
                    Write('Enter details for another student? (Y/N) ');  
                    Readln(Response);  
                end;  
            end;  
        end;  
    end;  
end;
```

**Q6 (b): Python**

```
def TopLevel() :  
    Response = "Y"           # string/character variable  
    InputData = ""           # string variable  
    Success = True           # Boolean variable  
    while Response == "Y" :  
        InputData = GetInfo()  
        if InputData[0] < "N" :  
            Success = WriteInfo(InputData, "File1.txt")  
        else :  
            Success = WriteInfo(InputData, "File2.txt")  
        if not Success :  
            Response = "Y"  
        else :  
            Response = input("Enter details for another student? (Y/N) ")
```

QUESTION 14.



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 15.



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 <code>Index</code> and used to index array <code>Duplicates</code> 2. OPEN file <code>UserListAtoM.txt</code> in READ mode and CLOSE 3. Pre-Condition loop to go through the file until <code>EOF()</code> and early termination if array full 4. Read line from file and extract account number (<code>AccNum</code>) in a loop 5. Call <code>SearchFileNtoZ</code> (with <code>AccNum</code>) following an attempt at MP4 in a loop 6. Check if return value is TRUE and if so: in a loop 7. store <code>AccNum</code> in correct array element 8. increment array <code>index</code> 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 16.



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)
```