QUESTION 2.

1 (a)

Identifier	Data Type	Description
`RaceHours`	`INTEGER`	The hours part of the race time
`RaceMinutes`	`INTEGER`	the minute part of the race time
`RaceSeconds`	`INTEGER // REAL`	the seconds part of the race time
`RaceTime`	`INTEGER // REAL`	the race time in seconds

3 × (meaningful name + data type) [3]

(b) (i)

Identifier	Data Type	Description
`PersonalBestTime`	`INTEGER/REAL`	Personal best time in seconds

meaningful name + data type [1]

(ii) Mark as follows:

- Declarations/comments for variables – at least 2
- Input (+ prompts) for hours, minutes, seconds
- Input (+ prompt) of personal best time
- Correct calculation of `RaceTimeInSeconds` (don't allow use of 'x' for '*')
- Output `RaceTimeInSeconds`
- Correct logic and output message for < personal best
- Correct logic and output message for > personal best
- Correct logic and output message for = personal best [max 7]

(c) (i)

- Choosing data/values…
- Test every possible 'logic path' through the code
  // with knowledge of the structure/code

Ignore any reference to normal/boundary/extreme … [2]

(ii)

- `PersonalBest` column labelled
- Test number 1 message: "Equals personal best time"/or similar
- Test 2/Test 3 – data for better performance …
- Described with suitable message
- Test 2/Test 3 – data for worse performance …
- Described with suitable message [6]

2 (a) (i) Displays the menu (choices)

QUESTION 3.

Question	Answer	

6(a)	Pseudocode solution included here for development and clarification of mark scheme.
	Programming language solutions appear in the Appendix.
	
	`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`
	`ENDFOR`
	
	`IF Not (LCaseChar>=2 AND UCaseChar>= 2 AND NumChar>= 3)`
	`  THEN`
	`  ReturnFlag ← FALSE`
	`ENDIF`
	`RETURN (ReturnFlag)`
	`ENDFUNCTION`
	
	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. "AAbb123")**
	
	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

   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 4.

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.

Generic Marking Principles

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

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.

<table>
<tr><th>Question</th><th>Answer</th><th></th></tr>
<tr><td>1(a)(i)</td><td>
<table>
<tr><th>Statement</th><th>Selection</th><th>Repetition (Iteration)</th><th>Assignment</th></tr>
<tr><td>WHILE Count < 20</td><td></td><td>✓</td><td></td></tr>
<tr><td>Count ← Count + 1</td><td></td><td></td><td>✓</td></tr>
<tr><td>If MyGrade <> 'C' THEN</td><td>✓</td><td></td><td></td></tr>
<tr><td>Mark[Count] ← GetMark(StudentID)</td><td></td><td></td><td>✓</td></tr>
<tr><td>ELSE OUTPUT "Fail"</td><td>✓</td><td></td><td></td></tr>
<tr><td>ENDFOR</td><td></td><td>✓</td><td></td></tr>
</table>
One mark for each row
</td><td></td></tr>
<tr><td>1(b)(i)</td><td>
<table>
<tr><th>Statement</th><th>Data type</th></tr>
<tr><td>MyAverage ← 13.5</td><td>REAL</td></tr>
<tr><td>ProjectCompleted ← TRUE</td><td>BOOLEAN</td></tr>
<tr><td>Subject ← "Home Economics"</td><td>STRING</td></tr>
<tr><td>MyMark ← 270</td><td>INTEGER</td></tr>
<tr><td>MyGrade ← 'B'</td><td>CHAR</td></tr>
</table>
</td><td>5</td></tr>
<tr><td>1(b)(ii)</td><td>
<table>
<tr><th>Expression</th><th>Evaluates to</th></tr>
<tr><td>"Air-" & MID(Subject, 7, 3)</td><td>"Air-con"</td></tr>
<tr><td>INT(MyAverage / 2)</td><td>6</td></tr>
<tr><td>ProjectCompleted AND MyMark > 270</td><td>FALSE</td></tr>
<tr><td>ProjectCompleted OR MyMark > 260</td><td>TRUE</td></tr>
<tr><td>ASC(MyGrade / 3)</td><td>ERROR</td></tr>
</table>
</td><td>5</td></tr>
</table>

Question	Answer	

2(a)	`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`
	
	`  IF Today() = 3`
	`  THEN`
	`  DRate ← DRate * 1.2`
	`  ENDIF`
	
	`  RETURN DRate`
	
	`ENDFUNCTION`
	
	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: / Dynamic Syntax Checking Explanation: Highlights the error as code is typed in	**2**

<table>
<tr><th>Question</th><th>Answer</th></tr>
<tr><td>3(a)</td><td>Code has to be in machine code (or equivalent) to be executed</td></tr>
<tr><td>3(b)</td><td>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</td></tr>
<tr><td>3(c)</td><td><table>
<tr><th>Statement</th><th>White-box</th><th>Black-box</th></tr>
<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>
</table>
1 mark per row</td><td>4</td></tr>
</table>

<table>
<tr><th>Question</th><th>Answer</th><th></th></tr>
<tr><td>4(a)(i)</td><td>
<table>
<tr><td>The identifier name of a global integer referenced</td><td>NumElements</td></tr>
<tr><td>The identifier name of a user-defined procedure</td><td>SaveToFile</td></tr>
<tr><td>The line number of an unnecessary statement</td><td>16</td></tr>
<tr><td>The scope of ArrayString</td><td>Local</td></tr>
</table>
</td><td></td></tr>
<tr><td>4(a)(ii)</td><td>1 mark for each mark point:

• Loop / repeat / iterate through array ResultArray one element at a time
• extract a string from row / column 1 of the array
• compare the string with SearchString
• if they match, call SaveToFile() and increment NumberFound</td><td>4</td></tr>
<tr><td>4(b)</td><td>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:

1 Function header and end including parameter and return
2 Declaration of two local variables as above but NOT NumElements
3 FOR … ENDFOR loop with range as given
4 Referencing each element from the array
5 Converting both strings to uppercase / lowercase
6 If strings are equal then Call SaveToFile() and increment NumberFound</td><td>6</td></tr>
</table>

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.
	
	`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`
	
	One mark for:
	• `ResultArray` declaration / commented in Python
	• assigning to all elements
	• assignment of '*'
		3

Question	Answer
5	

```
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
```

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 `NULL`s
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;

    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.
```

Q4 (d): Python

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

    ResultArray = [[0] * 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 5.

Question	Answer	

2(a)(i)	`FUNCTION CalcPoints(CardNum: STRING, Total: REAL) RETURNS INTEGER`
	`  DECLARE OldPoints : INTEGER`
	`  DECLARE NewPoints : INTEGER`
	
	`  IF Total > 100`
	`  THEN`
	`  OldPoints ← GetPoints(CardNum)`
	`  IF OldPoints > 2000`
	`  THEN`
	`  NewPoints ← INT(Total * 1.2)`
	`  ELSE`
	`  NewPoints ← INT(Total * 1.1)`
	
	`  ENDIF`
	`  ELSE`
	`  NewPoints ← 0`
	`  ENDIF`
	
	`  RETURN NewPoints`
	
	`ENDFUNCTION`
	
	1 mark for each of the following:
	
	1 Correct `FUNCTION` heading (as given) and end
	2 Declaring local variables for `OldPoints` and `NewPoints`
	3 `IF...THEN...ELSE...ENDIF` with `Total > 100`
	4 …Newpoints set to zero if Total <= 100
	5 Nested `IF...THEN...ELSE...ENDIF` with `OldPoints > 2000`
	6 … with correct assignments of NewPoints
	7 Return `NewPoints` for all cases	

Question	Answer	

2(a)(ii)	`FUNCTION GetTotal() RETURNS REAL`
	
	`  DECLARE Valid : BOOLEAN`
	`  DECLARE Amount : REAL`
	`  Valid ← FALSE`
	
	`  REPEAT`
	`  OUTPUT "Enter the amount"`
	`  INPUT Amount`
	`  IF Amount > 0 AND Amount < 10000`
	`  THEN`
	`  Valid ← TRUE`
	`  ENDIF`
	`  UNTIL Valid = TRUE`
	
	`  RETURN Amount`
	
	`ENDFUNCTION`
	
	Note that the pseudocode shown is only an example. The use of an explicit flag and `IF` structure are not essential provided the functionality is provided.
	
	1 mark for each of:
	
	1 declaration of local variable(s) used
	2 prompt followed by input
	3 conditional loop
	4 checking that input value > 0 and input value < 10000
	5 returning the value	

2(b)(i)	1 mark for name, 1 mark for description Accept by example for description Name: Run-time Description: The program executes an illegal instruction // performs an illegal operation that is trapped by the OS	**2**
2(b)(ii)	One **specific value** from within each of the following ranges: For example: • 73 (a value <= 100) • 145.3 (a value > 100) 1 mark for the value, plus one for a meaningful description.	**4**

QUESTION 6.

Question	Answer	

2(a)(i)	`FUNCTION CalcBonus(CardNum: STRING) RETURNS INTEGER`
	
	`DECLARE Points : INTEGER`
	`DECLARE Bonus : INTEGER`
	`DECLARE Spend : REAL`
	
	`Points ← GetPoints(CardNum)`
	`Spend ← GetSpend(CardNum)`
	
	`IF Points > 2000`
	`THEN`
	`Bonus ← 100`
	`ELSE`
	`IF Spend > 1000`
	`THEN`
	`Bonus ← 50`
	`ELSE`
	`Bonus ← 10`
	`ENDIF`
	`ENDIF`
	
	`RETURN Bonus`
	
	`ENDFUNCTION`
	
	1 mark for each of the following up to max 5 marks:
	
	1 Function heading (inc parameters) **and** ending
	2 Declaring local variables and two function calls as above
	3 `IF…THEN…ELSE…ENDIF` with `Points > 2000`
	4 (Nested) `IF…THEN…ELSE` with `Spend > 1000`
	5 … assignment of `Bonus` to 10, 50 100
	6 Return parameter	

Question	Answer	

2(a)(ii)	The pseudocode shown here is only an example. The use of an explicit flag and IF structure are not essential provided the functionality is provided.
	
	`FUNCTION GetCardNumber() RETURNS STRING`
	
	`DECLARE Valid : BOOLEAN`
	`DECLARE CardNum : STRING`
	
	`Valid ← FALSE`
	
	`REPEAT`
	`OUTPUT "Enter card number"`
	`INPUT CardNum`
	`IF LENGTH(CardNum) = 16 AND IS_NUM(CardNum) = TRUE`
	`THEN`
	`Valid ← TRUE`
	`ENDIF`
	`UNTIL Valid`
	
	`RETURN CardNum`
	
	`ENDFUNCTION`
	
	1 mark for each of the following:
	
	1 Declaring local variable to store user input
	2 Conditional Loop
	3 Prompt and input of `CardNum`
	4 Length check
	5 Checking `IS_NUM(CardNum)` is TRUE
	6 Return a value	

2(b)(i)	Name: Logic (error) Description: Where the program does not behave as expected / Does not give expected result / An error in the logic of the algorithm **OR** Name: Run-time // execution (error) Description: The program performs an illegal operation One mark for name + one mark for corresponding description	**2**
2(b)(ii)	Values: any `Spend` value and `Points` > 2000 Justification: `Bonus` should be 100 Values: `Spend` > 1000 and `Points` <= 2000 Justification: `Bonus` should be 50 Values: `Spend` <= 1000 and `Points` <= 2000 Justification: `Bonus` should be 10 2 marks for values 2 marks for relevant and appropriate reasons	**4**

Question	Answer	
2(c)	Name: Corrective Reason: Amend the algorithm to 'eliminate errors' Name: Adaptive Reason: In response to specification change arising from changes to business rules or environment (regulatory) Name: Perfective Reason: To make improvements to the program One mark for each name plus one mark for corresponding reason up to max 4 marks	

QUESTION 7.

<table>
<tr><th>Question</th><th>Answer</th><th></th></tr>
<tr><td>1(a)(i)</td><td>Construct: Assignment
Pseudocode example: <code>Answer ← "YES"</code>

Construct: Selection
Pseudocode example: <code>IF X = 3 THEN OUTPUT "HELLO"</code>

Construct: Repetition / Iteration
Pseudocode example: <code>FOR N ← 1 to 100</code>

One mark for construct
One mark for pseudocode example
Maximum 4 marks</td><td></td></tr>
<tr><td>1(a)(ii)</td><td>
<table>
<tr><th>Pseudocode statement</th><th>Input</th><th>Process</th><th>Output</th></tr>
<tr><td><code>Temp ← SensorValue * Factor</code></td><td></td><td>✓</td><td></td></tr>
<tr><td><code>WRITEFILE "LogFile.txt", TextLine</code></td><td></td><td></td><td>✓</td></tr>
<tr><td><code>WRITEFILE "LogFile.txt", MyName & MyIDNumber</code></td><td></td><td>✓</td><td>✓</td></tr>
<tr><td><code>READFILE "AddressBook.txt", NextLine</code></td><td>✓</td><td>(✓)</td><td></td></tr>
</table>

One mark per correct row</td><td>4</td></tr>
<tr><td>1(b)(i)</td><td>
<table>
<tr><th>Expression</th><th>Evaluates to</th></tr>
<tr><td><code>MID(Title, 5, 3) & RIGHT(Author, 3)</code></td><td>"tripod"</td></tr>
<tr><td><code>INT(WeightEach * PackSize)</code></td><td>24</td></tr>
<tr><td><code>PackSize >= 4 AND WeightEach < 6.2</code></td><td>FALSE</td></tr>
<tr><td><code>LEFT(Author, ASC(Version) - 65)</code></td><td>"Er"</td></tr>
<tr><td><code>RIGHT(Title, (LEN(Author) - 6))</code></td><td>"hetti"</td></tr>
</table>

Quotes must be present
Must be capital E in row 4</td><td>5</td></tr>
</table>

Question	Answer	
1(b)(ii)	<table><tr><th>Variable</th><th>Data type</th></tr><tr><td>Tile</td><td>STRING</td></tr><tr><td>Version</td><td>CHAR</td></tr><tr><td>PackSize</td><td>INTEGER</td></tr><tr><td>WeightEach</td><td>REAL</td></tr><tr><td>Paperback</td><td>BOOLEAN</td></tr></table> One mark per data type	
1(c)	Data is chosen: • to test that the program does what it is supposed to do / to check that the results are as expected • to use known valid, boundary and erroneous values	**2**

QUESTION 8.

Question	Answer	Marks
5(a)(i)	To test <u>every path</u> through the code / algorithm Accept phrase with equivalent meaning	**1**
5(a)(ii)	A trace table	**1**
5(b)(i)	**String:** • three possible formats for string containg two words: `"Cat∇∇Dog" // "Cat∇Dog∇" // "∇Cat∇Dog"` **Explanation:** • When a space character is encountered • `NumWords` is incremented by 1 **OR:** • The algorithm counts the spaces • and not the words 1 mark for a string that would give the correct result 2 marks for explanation	**3**

Question	Answer	
5(b)(ii)	**String 1:** • Output: `Number of words : 2` **Description 1:** • Check character at end of string • If not a character, increment variable `NumTotal` **OR:** • Add a space at the beginning / end of string • At the start of the algorithm **OR:** • Change Initialisation of `NumWords` • to 1 **OR:** • After the loop • Add 1 to `NumWords` **String 2:** • Output: `Number of words : 5` **Description 2:** • Detect a space followed by a space • Count as a single space / only increment variable `NumTotal` once **OR:** • Replace all double spaces with a single space • Before the loop **OR:** • After the loop • Subtract 2 from `NumWords` Many possible solutions. One mark for each correct output One marks for each description bullet point	

QUESTION 9.

Question	Answer	

4(a)	
	`PROCEDURE Button(ButtonNum : INTEGER)`
	`  DECLARE Limit : INTEGER`
	
	`  IF ButtonNum = 10 // increase volume`
	`  THEN`
	`  IF MaxVol = 0`
	`  THEN`
	`  Limit ← 49`
	`  ELSE`
	`  Limit ← MaxVol`
	`  ENDIF`
	`  IF VolLevel < Limit`
	`  THEN`
	`  VolLevel ← VolLevel + 1`
	`  ENDIF`
	`  ELSE // otherwise must be ButtonNum 20 - decrease`
	`  IF VolLevel > 0`
	`  THEN`
	`  VolLevel ← VolLevel - 1`
	`  ENDIF`
	`  ENDIF`
	`ENDPROCEDURE`
	
	Mark as follows:
	
	1. Check if parameter value = 10
	2. Check if parameter value = 20
	3. Check if `MaxVol = 0`
	4. Decrement `VolLevel` **and** ensure still in range
	
	If attempting to increase volume (parameter value was 10):
	
	5. Increment `VolLevel`
	6. Ensure `VolLevel` still in range: for both cases. i.e:
	`VolLevel <= 49` (for `MaxVol = 0`)
	`VolLevel <= MaxVol` (for `MaxVol <> 0`)	

4(b)	2 independent marks for each test:
	
	Test 1
	`MaxVol: 0/49`
	`VolLevel expected: 49`
	
	Test 2
	`VolLevel before: 34`
	`VolLevel expected: 34`
	
	Test 3
	`Parameter: 20`
	`VolLevel expected: 0`
		6

Question	Answer	
4(c)(i)	One mark for type, one for description: • Type: Logical Error • Description: Program does not perform as expected • Type: Run-time error • Description: Program executes an invalid instruction / out of bounds error / attempts to divide by zero // program crashes	
4(c)(ii)	One mark per bullet point: • Tests carried out before all the modules / subroutines have been written • Simple / dummy module written to simulate / model / replace the actual module / subroutine / object • Contains an output statement // returns a fixed value to indicate that the call has been made	**3**