Cambridge Assessment International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9608/22**

Paper 2 Fundamental Problem-solving and Programming Skills **May/June 2021**

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 2021 series for most Cambridge IGCSE™, Cambridge International A and AS Level components and some Cambridge O Level components.

This document consists of **20** printed pages.

© UCLES 2021

Generic Marking Principles

These general marking principles must be applied by all examiners when marking candidate answers. 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.

© UCLES 2021

Mechanics of Marking:

Every mark given should have a corresponding tick on the script.

Every part question must be annotated to show that it has been read.

There are <u>four</u> pages that appear at the start of the script (including the Appendix page) that must be annotated with the SEEN icon. The easiest way to do this is to go the first question and select Zoom > 'fit height' then all six can be annotated at the same time without scrolling.

Do not put comments on the scripts. When scripts are returned to centres all the annotations including comments, are visible.

If work has been crossed out and something written in its place, the replacement work is marked even if the crossed-out work is correct. If the crossed-out work has not been replaced, mark the crossed-out answer.

For single mark answers, mark the first answer on the line, unless there is a note to the contrary on the mark scheme.

If a candidate writes something that is not enough (NE) for a mark, but is not actually incorrect, continue reading, even if the mark scheme says, for example, mark first two answers.

The use of NR (No Response) is described in this extract from the RM Assessor guide:

(d) Correct use of No Response (NR) and Zero (0) marks

Team Leaders should check that No Response is being used correctly by examiners.

(i) Award No Response (NR):

- if there is nothing written at all in the answer space, or
- if there is only a comment which does not in any way relate to the question being asked (e.g. 'can't do', 'don't know'), or
- if there is any sort of mark which is not an attempt at the question (e.g. a dash, a question mark).

Note: you can press the # or / key to enter NR.

(ii) Award Zero (0):

- if there is any attempt that earns no credit. This could, for example, include the candidate copying all or some of the question, or any working that does not earn any marks, whether crossed out or not.

For questions requiring program code, if the only thing that is written is the name of the program language then award NE.

Annotation requirement for multi-page responses:

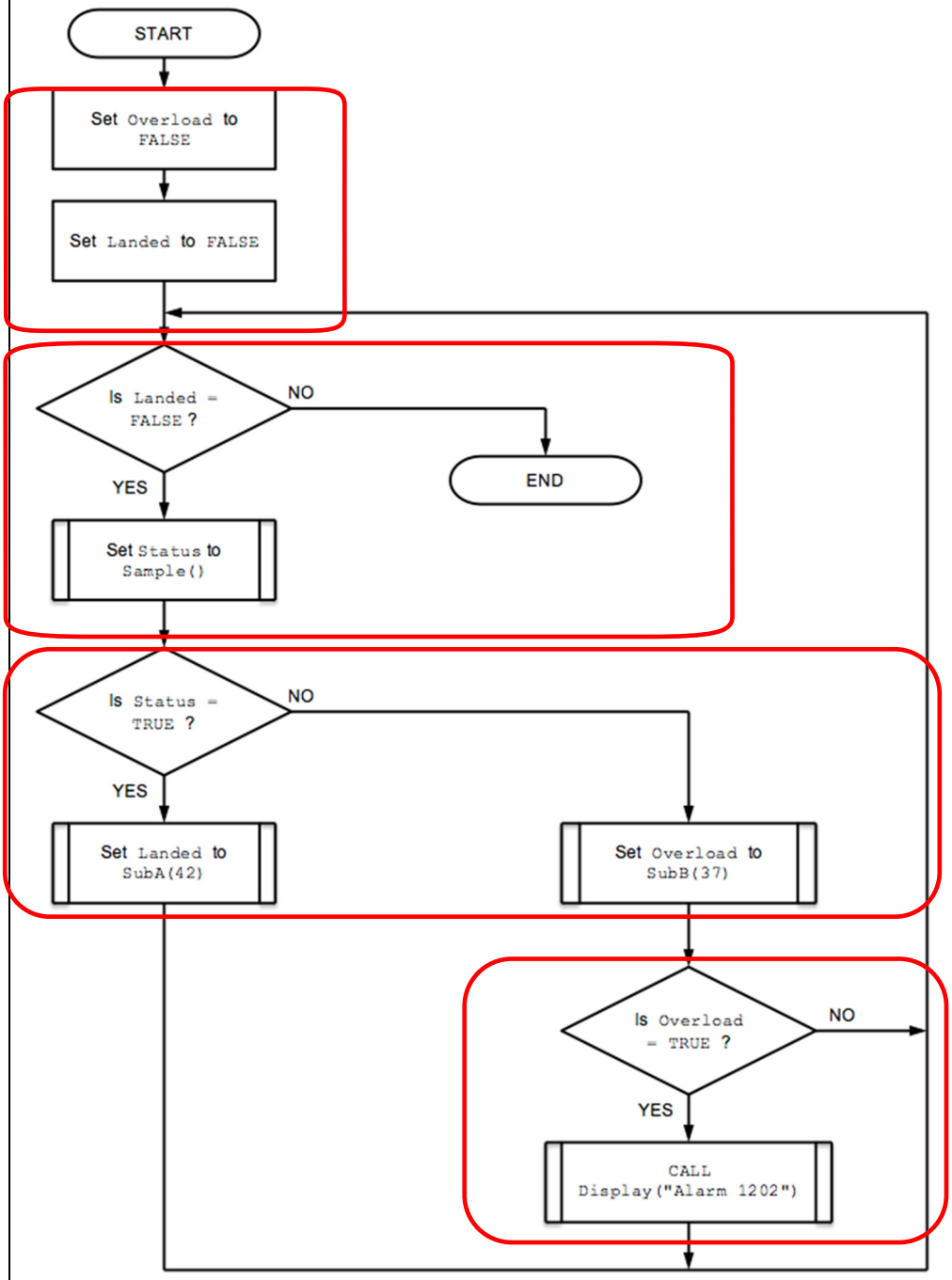
Question 6(b) and Question 6(c)

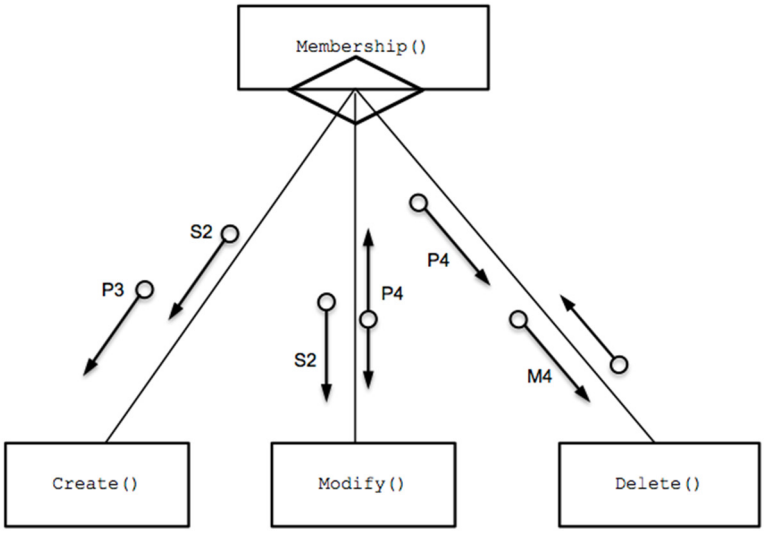
In each case, if the second page of the response (page 15 and page 17 respectively) is blank then add the annotation 'SEEN' to the second page.

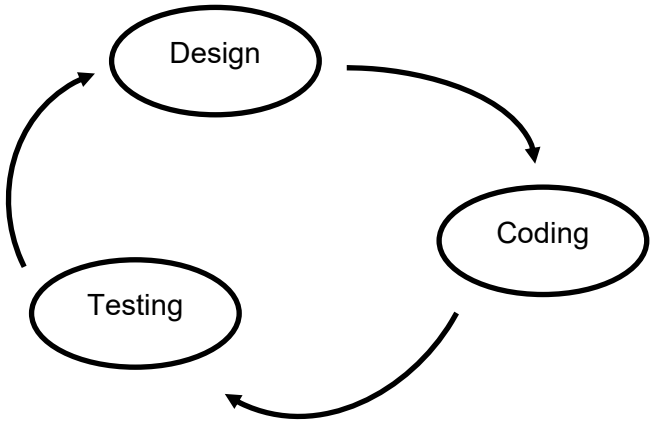
Question	Answer	Marks
1(a)	One mark for each type **and** one mark for corresponding reason (not dependant) Type: Corrective Reason: Because the program does not function as intended / contains a bug Type: Adaptive Reason: Change the program due to a requirement / specification / legislative change Type: Perfective Reason: Improve the performance of the program / to enhance the program Type: Preventive Reason: Improve maintainability or reliability	**4**
1(b)	One mark for: • Program / computer / CPU can only process / store <u>binary</u> values. • Standard encoding recognised by all programs / used to exchange information **Note: Max 1 mark**	**1**
1(c)	One mark per bullet point: • To provide a <u>separator</u> (or implied) between the data items • Algorithm to extract / locate individual items from a line of text is simplified • The special character does not appear in the data **Note: Max 2 marks**	**2**

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>1(d)</td><td>
<table>
<tr><th>Statement</th><th>Error</th></tr>
<tr><td>Code ← RIGHT("Cap" & "art", 4)</td><td>NO ERROR</td></tr>
<tr><td>Status ← MID("Computer", 7, 5)</td><td>Not enough characters</td></tr>
<tr><td>Size ← LENGTH("Password") * 2</td><td>NO ERROR</td></tr>
<tr><td>NextChar ← CHR('A')</td><td>CHR() param should be integer // CHR() should be ASC()</td></tr>
<tr><td>Index ← Index & 3</td><td>3 is not character (should be ‘3’) // and is not an arithmetic operator</td></tr>
</table>
One mark for each line
</td><td>5</td></tr>
</table>

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>2(a)</td><td>
<table>
<tr><th></th><th>Answer</th></tr>
<tr><td>The identifier name of a global variable</td><td>Overload</td></tr>
<tr><td>The name of the loop structure</td><td>Pre-condition loop</td></tr>
<tr><td>The identifier involved in a type mismatch</td><td>Landed</td></tr>
<tr><td>The name of a procedure that takes a parameter</td><td>Display()</td></tr>
<tr><td>The name of a function</td><td>Sample() // SubA() // SubB()</td></tr>
</table>
One mark per row
</td><td>5</td></tr>
</table>

Question	Answer	Marks
2(b)		5

Question	Answer	Marks
3(a)(i)	One mark for each: Module... • Hierarchy / relationships • Selection • Repetition / Iteration • Sequence **Note: Max 2 marks**	**2**
3(a)(ii)	 One mark for each of: 1 Diagram with all boxes correctly labelled, positioned as shown 2 Selection diamond as shown 3 (P3 **and** S2) and (P4 **and** M4) *(Parameters to Create and Delete)* 4 S2 **and** P4 (double arrow) *(Parameters to Modify)* 5 Return parameter from `Delete()`	**5**

Question	Answer	Marks
3(b)	 One mark for: • Design, Coding and Testing (in sequence) • Arrows as shown // Alternative 'waterfall model' where each stage loops back to the previous stage	**2**

Question	Answer	Marks

4(a)	`DECLARE Num : INTEGER`
	`Num ← 101`
	
	`REPEAT`
	`  OUTPUT Num`
	`  Num ← Num + 2`
	`UNTIL Num > 199`
	
	Mark as follows:
	1 Counter declaration and initialisation to sensible value **before the loop**
	2 `REPEAT ... UNTIL <condition>`
	3 Correct selection of number to be output (use of `MOD()` or `+2` or other)
	4 Correct range of numbers output
		4

Question	Answer	Marks

4(b)	'Pseudocode' solution included here for development and clarification of mark scheme.
	Programming language example solutions appear in the Appendix.
	
	`FUNCTION Search(Par1, Par2 : STRING) RETURNS INTEGER`
	
	`DECLARE Index, RetVal : INTEGER`
	
	`Index ← 1`
	`RetVal ← -1`
	
	`WHILE Index <= 100 AND RetVal = -1`
	`  IF XRef[Index, 1] = Par1 _`
	`  AND (XRef[Index, 2] = Par2 OR XRef[Index, 3] = Par2)`
	`  THEN`
	`  RetVal ← Index`
	`  ENDIF`
	`  Index ← Index + 1`
	`ENDWHILE`
	
	`RETURN RetVal`
	`ENDFUNCTION`
	
	Mark as follows:
	1 Function heading and ending including parameters
	2 Declaration of local variable for array index (`Index`) but **not** of global `XRef` array
	3 Any loop for 100 elements
	4 Use of correct `XRef` 2D array syntax **in a loop**
	5 Logical test of array elements **in a loop**
	6 Exit loop if match found (following attempt at MP5)
	Return index number or −1 if not found
		7

Question	Answer	Marks
5(a)(i)	One mark per point: • Pretty print / Colour-coding of keywords / variables • (Auto) indentation • (Auto) Expansion / collapsing of data structures / code blocks // thumbnail overview **Max 2 marks**	**2**
5(a)(ii)	One mark per point: • Dynamic syntax checking / highlighting syntax errors as code is typed • Type checking • Checking for used variables not declared / unused variables which are declared **Max 2 marks**	**2**

Question	Answer	Marks
5(b)(i)	**DECLARE Response : BOOLEAN** DECLARE Password : STRING OUTPUT "Please Input your password: " INPUT Password Response ←Validate(UserID, Password) AND Today() **RETURN Response** One mark per line in **bold**	**2**

5(b)(ii)	'Pseudocode' solution included here for development and clarification of mark scheme.
	Programming language example solutions appear in the Appendix.
	
	FUNCTION Verify(UserID : STRING) RETURNS BOOLEAN
	  DECLARE Count : INTEGER
	  DECLARE Response : BOOLEAN
	  DECLARE Password : STRING
	
	  Count ← 1
	  Response ← FALSE
	
	  IF UserID = "Guest" THEN
	  Response ← TRUE
	  ENDIF
	
	  WHILE Response = FALSE AND Count < 4
	  OUTPUT "Please input your password: "
	  INPUT Password
	  Response ← Validate(UserID, Password) AND TODAY()
	  IF Response = FALSE THEN
	  IF Count < 3 THEN
	  OUTPUT "Incorrect password - please try again"
	  ENDIF
	  Count ← Count + 1
	  ENDIF
	  ENDWHILE
	
	  RETURN Response
	ENDFUNCTION
	
	1 mark for each of the following:
	
	1 If <u>parameter</u> matches "Guest" then **skip** to Return (MP7)
	2 Loop for max 3 attempts and terminate if valid password input
	3 Prompt and Input first password attempt
	4 Evaluate result of `Validate() AND TODAY()` **in a loop**...
	5 Test result and increment count for incorrect password **in a loop**...
	6 ...output 'try again' message and re-input password **in a loop**...
	7 Return Boolean (correctly in all 3 cases)
	
	Note: Max 6 marks
		6

Question	Answer	Marks

6(a)	'Pseudocode' solution included here for development and clarification of mark scheme.
	Programming language example solutions appear in the Appendix.
	
	`PROCEDURE StockSummary()`
	`  DECLARE Index, Count : INTEGER`
	`  DECLARE Total : REAL`
	
	`  Count ← 0`
	`  Total ← 0.0`
	
	`  FOR Index ← 1 TO 10000`
	`  IF StockID[Index]) <> "" THEN`
	`  Count ← Count + 1`
	`  Total ← Total + (Quantity[Index] * Cost[Index])`
	`  ENDIF`
	`  ENDFOR`
	`  OUTPUT "Total Value is ", Total`
	`  OUTPUT "Number of Stock Entries is ", Count`
	
	`ENDPROCEDURE`
	
	One mark for each of the following:
	
	1 Declaration **and** initialisation of `Count` and `Total` (by comment in Python)
	2 Loop for 10000 elements
	3 Skip row when `StockID` element = "" **in a loop**
	4 Increment `Count` and sum `Total` **in a loop**
	5 `OUTPUT` the total and the count **after the loop** (following a reasonable attempt at MP4)
		5

Question	Answer	Marks
6(b)		**8**

```
FUNCTION Restore(Filename : STRING) RETURNS BOOLEAN

  DECLARE Index : INTEGER
  DECLARE FileLine : STRING
  DECLARE Success : BOOLEAN

  Success ← TRUE
  OPENFILE Filename FOR READ
  READFILE Filename, FileLine
  IF FileLine = "" THEN       //alt: IF EOF(filename)
     CLOSEFILE Filename
     RETURN FALSE             //file is empty
  ENDIF

  FOR Index ← 1 TO 10000      //first initialise arrays
     StockID[Index] ← ""
     Description[Index] ← ""
     Quantity[Index] ← 0
     Cost[Index] ← 0.0 // 0
  ENDFOR

  Index ← 1          //starts loop with first FileLine
  WHILE NOT EOF(Filename) AND Index <= 10000
     CALL Unpack(Index, FileLine)
     Index ← Index + 1
     READFILE Filename, FileLine
  ENDWHILE

  IF Index = 10001 AND NOT EOF(Filename) THEN
     Success ← FALSE
  ENDIF
  CLOSEFILE Filename

  RETURN Success

ENDFUNCTION
```

1 mark for each of the following:

1 Function heading including input parameter **and** function End
2 Declare local variable for line read from file
3 `OPEN` file in READ mode **and** `CLOSE`
4 Check whether file is empty **and** return `FALSE` if it is (no restore
5 Loop through all 10 000 elements initialising arrays
6 Loop until EOF(Filename) OR Index > 10000
7 Call `Unpack()` correctly for each line from the file **in a loop**
8 Return `FALSE` if more than 10000 lines in file, otherwise returns `TRUE`

Question	Answer	Marks

6(c)	'Pseudocode' solution included here for development and clarification of mark scheme.
	Programming language example solutions appear in the Appendix.
	
	`FUNCTION GetValidFilename() RETURNS STRING`
	`  DECLARE Filename : STRING`
	`  DECLARE Valid : BOOLEAN`
	`  DECLARE Index : INTEGER`
	`  DECLARE NextChar : CHAR`
	
	`  REPEAT`
	`  Valid ← TRUE`
	`  OUTPUT "Please input the name for the backup file "`
	`  INPUT Filename`
	`  IF LENGTH(Filename) < 4 OR LENGTH(Filename) > 10`
	`  THEN`
	`  Valid ← FALSE`
	`  ELSE`
	`  Index ← 1`
	`  WHILE Index <= LENGTH(Filename) AND Valid = TRUE`
	`  NextChar ←  MID(Filename, Index, 1)`
	`  IF NOT ((NextChar >='a' AND NextChar <='z')`
	`  OR  (NextChar >='A' AND NextChar <='Z')`
	`  OR  (NextChar >='0' AND NextChar <='9'))`
	`  THEN`
	`  Valid ← FALSE //not alphanumeric`
	`  ENDIF`
	`  Index ← Index + 1`
	`  ENDWHILE`
	`  ENDIF`
	
	`  IF Valid = FALSE THEN`
	`  OUTPUT "Invalid filename - please try again"`
	`  ENDIF`
	
	`  UNTIL Valid = TRUE`
	`  RETURN Filename`
	`ENDFUNCTION`
	
	One mark for each of the following:
	1 Conditional loop until valid filename input
	2 Prompt and Input of filename **in a loop**
	3 Test length is within range
	4 Loop through filename:
	5 Extract a single character
	6 Test that character is numeric **in a loop**
	7 Test that character is alphabetic **in a loop**
	8 If filename invalid, output warning and repeat, if valid then return filename
		8

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

Program Code Example Solutions

Question 4(b): Visual Basic

```
Function Search(Par1, Par2 As STRING) As INTEGER

Dim Index, RetVal As INTEGER

Index = 1
RetVal = -1

While Index <= 100 And RetVal = -1
   If XRef(Index, 1) = Par1 _
         AND (XRef(Index, 2) = Par2 OR XRef(Index, 3) = Par2) Then
         RetVal = Index
   End If
   Index = Index + 1
End While

Return RetVal
End Function
```

Question 4(b): Pascal

```
function Search(Par1, Par2 : string) : integer;

var
  Index : integer;
  RetVal : integer;

begin

  Index := 1;
  RetVal := -1;

  While Index <= 100 And RetVal = -1 do
  begin
     if XRef[Index, 1] = Par1
        and (XRef[Index, 2] = Par2 or XRef[Index, 3] = Par2) then
        RetVal := Index;

     Index := Index + 1;
  end;

  Search := RetVal // result := RetVal
end;
```

Question 4(b): Python

```
def Search(Par1, Par2):

  ## Index, RetVal As INTEGER

  Index = 1
  RetVal = -1

  while Index <= 100 and RetVal = -1:
     if XRef[Index][1] == Par1 \
         and XRef[Index][2] == Par2 or XRef[Index][3] == Par2:
         RetVal = Index
     Index = Index + 1

  return RetVal
```

Question 5(b)(ii): Visual Basic

```
Function Verify(UserID As String) As Boolean
  Dim Count As INTEGER
  Dim Response As BOOLEAN
  Dim Password As STRING

  Count = 1
  Response = FALSE

  If UserID = "Guest" Then
        Return TRUE
  End If

  While Response = FALSE And Count < 4
     Console.Writeline("Please input your password: ")
     Password = Console.Readline()
     Response = Validate(UserID, Password) AND TODAY()
     If Response = FALSE Then
        Count = Count + 1
        If Count < 4 Then
           Console.Writeline("Incorrect password - please try again")
        End If
     End If
  End While

  Return Response

End Function
```

Question 5(b)(ii): Pascal

```
function Verify(UserID : string) : boolean;
var
   Count : integer;
   Response : boolean;
   Password : string;

begin
   Count := 1;

   if UserID = "Guest" then
      Verify := TRUE // result := TRUE;

   while Response = FALSE And Count < 4
   begin
      writeln('Please input your password: ');
      readln(Password);
      Response := Validate(UserID, Password) and TODAY();
      if Response = FALSE then
      begin
         Count := Count + 1;
         if Count < 4 then
            writeln('Incorrect password - please try again');
      end;
   end;

   Verify := Response // result :=  Response

end;
```

Question 5(b)(ii): Python

```
def Verify(UserID):
  ## Count As INTEGER
  ## Response As BOOLEAN
  ## Password As STRING

  Count = 1

  if UserID == "Guest":
        return TRUE

  while Response == FALSE and Count < 4:
     Password = input("Please input your password: ")
     Response = Validate(UserID, Password) and TODAY()
     if Response == FALSE:
           Count = Count + 1
           if Count < 4:
              print("Incorrect password - please try again")

  return Response
```

Question 6(a): Visual Basic

```
Sub StockSummary()
  Dim Index, Count As Integer
  Dim Total As Real

  Count = 0
  Total = 0.0

  For Index = 1 To 10000
     If StockID(Index) <> "" then
           Count = Count + 1
           Total = Total + (Quantity(Index)) * Cost(Index))
     End If
  Next Index

  Console.writeline("Total Value is " & Total.toString)
  Console.writeline("Number of Stock Entries is " & Count.toString)

End Sub
```

Question 6(a): Pascal

```
procedure StockSummary();

var
  Index, Count : Integer;
  Total : Real;

begin
  Count := 0;
  Total := 0.0;

  for Index := 1 TO 10000 do
  begin
     if StockID[Index] <> "" then
     begin
        Count := Count + 1;
        Total := Total + (Quantity[Index]) * Cost[Index]);
     end;
  end;


  writeln('Total Value is ', Total);
  writeln('Number of Stock Entries is ', Count);

end;
```

Question 6(a): Python

```
def StockSummary():

  ## Index, Count : Integer
  ## Total : Real

  Count = 0
  Total = 0.0

   for Index in range(1, 10001):
      if StockID[Index] <> "":
         Count = Count + 1
         Total = Total + (Quantity[Index]) * Cost[Index])

    print("Total Value is ", Total)
    print("Number of Stock Entries is ", Count)
```

Question 6(c): Visual Basic

```
Function GetValidFilename() As String
  Dim Filename As String
  Dim Valid As Boolean
  Dim Index As Integer
  Dim NextChar as Char

  Valid = FALSE

  Do
     Valid = TRUE
     Console.writeline("Please input the name for the backup file ")
     Filename = Console.readline()
     If Len(Filename) < 4 Or Len(Filename) > 10 Then
        Valid = FALSE
     Else
        Index = 1
        While Index <= Len(Filename) And Valid = TRUE
           NextChar =  MID(Filename, Index, 1)
              If Not ((NextChar >='a' AND NextChar <='z') _
                 Or  (NextChar >='A' AND NextChar <='Z')_
                 Or  (NextChar >='0' AND NextChar <='9')) Then
                 Valid = FALSE //not alphanumeric
              End If
           Index = Index + 1
        End While

     End If

     If Valid = FALSE Then
           Console.writeline("Invalid filename - please try again")
     End If

  Loop Until Valid = TRUE
  Return Filename
End Function
```

Question 6(c): Pascal

```
function GetValidFilename() : String;
var
   Filename : String;
   Valid : Boolean;
   Index : Integer;
   NextChar : Char;

begin
   Valid := FALSE;

   repeat
      Valid := TRUE;
      writeln('Please input the name for the backup file ');
      readln(Filename);
      if length(Filename) < 4 Or length(Filename) > 10 then
         Valid := FALSE
      else
         begin
            Index := 1;
               while Index <= length(Filename) And Valid = TRUE do
               begin
                  NextChar :=  MidStr(Filename, Index, 1);
                  if Not ((NextChar >='a' AND NextChar <='z')
                      Or  (NextChar >='A' AND NextChar <='Z'
                      Or  (NextChar >='0' AND NextChar <='9')) then
                         Valid := FALSE; //not alphanumeric

                  Index := Index + 1;
               end;
         end;

      If Valid = FALSE then
         writeln('Invalid filename - please try again');

   until Valid = TRUE;
   result := Filename; // GetValidFilename := Filename
end;
```

Question 6(c): Python

```
def GetValidFilename():
  ## Filename As String
  ## Valid As Boolean
  ## Index As Integer
  ## NextChar As Char

  Valid = FALSE

  while not Valid:
     Valid = TRUE
     Filename = input("Please input the name for the backup file ")
     if Len(Filename) < 4 or Len(Filename) > 10:
        Valid = FALSE
     else:
        Index = 0
        while Index <= Len(Filename) and Valid = TRUE:
           NextChar = FileName[Index]
           if not ((NextChar >= '0' and NextChar <= '9'
              or (NextChar >= 'a' and NextChar <= 'z')
              or (NextChar >= 'A' and NextChar <= 'Z'):
```

```
            Valid = FALSE

         Index = Index + 1

   if Valid == FALSE:
      print("Invalid filename - please try again")


return Filename
```