



## 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.

**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.

## 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 four 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)

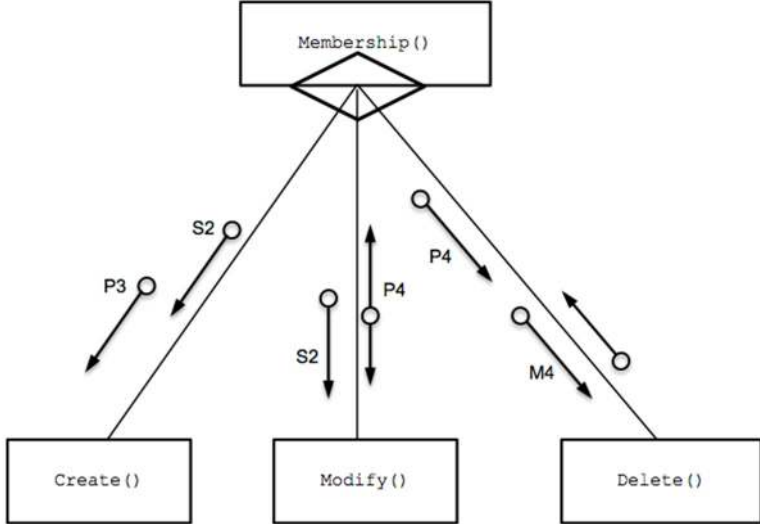
In each case, if the second page of the response (page 15 and page 17 respectively) is blank then add the annotation 'SEEN' to the second page.

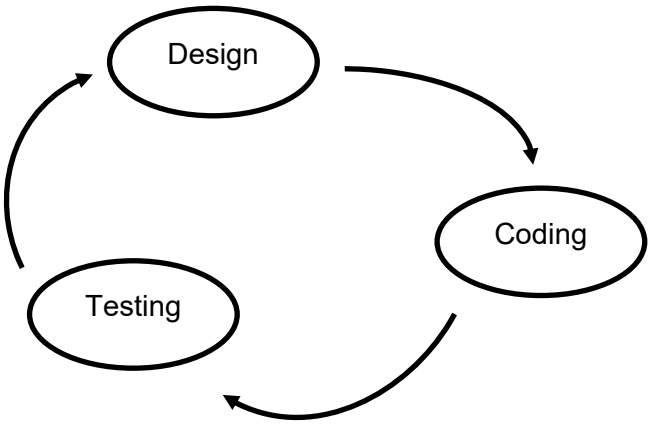
Question	Answer	Marks
1(a)	<p>One mark for each type <b>and</b> one mark for corresponding reason (not dependant)</p> <p>Type: Corrective Reason: Because the program does not function as intended / contains a bug</p> <p>Type: Adaptive Reason: Change the program due to a requirement / specification / legislative change</p> <p>Type: Perfective Reason: Improve the performance of the program / to enhance the program</p> <p>Type: Preventive Reason: Improve maintainability or reliability</p>	<b>4</b>
1(b)	<p>One mark for:</p> <ul style="list-style-type: none"> <li>• Program / computer / CPU can only process / store <u>binary</u> values.</li> <li>• Standard encoding recognised by all programs / used to exchange information</li> </ul> <p><b>Note: Max 1 mark</b></p>	<b>1</b>
1(c)	<p>One mark per bullet point:</p> <ul style="list-style-type: none"> <li>• To provide a <u>separator</u> (or implied) between the data items</li> <li>• Algorithm to extract / locate individual items from a line of text is simplified</li> <li>• The special character does not appear in the data</li> </ul> <p><b>Note: Max 2 marks</b></p>	<b>2</b>

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

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

Question	Answer	Marks
2(b)	<pre> graph TD     Start([START]) --&gt; S1[Set Overload to FALSE]     S1 --&gt; S2[Set Landed to FALSE]     S2 --&gt; D1{Is Landed = FALSE?}     D1 -- NO --&gt; End([END])     D1 -- YES --&gt; S3[Set Status to Sample()]     S3 --&gt; D2{Is Status = TRUE?}     D2 -- YES --&gt; S4[Set Landed to SubA(42)]     D2 -- NO --&gt; S5[Set Overload to SubB(37)]     S4 --&gt; D3{Is Overload = TRUE?}     S5 --&gt; D3     D3 -- YES --&gt; S6[CALL Display("Alarm 1202")]     D3 -- NO --&gt; D1     S6 --&gt; D1     </pre>	5

Question	Answer	Marks
3(a)(i)	<p>One mark for each:</p> <p>Module...</p> <ul style="list-style-type: none"> <li>• Hierarchy / relationships</li> <li>• Selection</li> <li>• Repetition / Iteration</li> <li>• Sequence</li> </ul> <p><b>Note: Max 2 marks</b></p>	<b>2</b>
3(a)(ii)	<div style="text-align: center;">  <pre> graph TD     subgraph MembershipBox [Membership()]         direction TB         Selection{ }     end     Create[Create()]     Modify[Modify()]     Delete[Delete()]          Selection -- P3 --&gt; Create     Selection -- S2 --&gt; Create     Selection -- S2 --&gt; Modify     Selection -- P4 --&gt; Modify     Selection -- P4 --&gt; Delete     Selection -- M4 --&gt; Delete          style Selection fill:none,stroke:none     </pre> </div> <p>One mark for each of:</p> <ol style="list-style-type: none"> <li>1 Diagram with all boxes correctly labelled, positioned as shown</li> <li>2 Selection diamond as shown</li> <li>3 (P3 <b>and</b> S2) and (P4 <b>and</b> M4) (<i>Parameters to Create and Delete</i>)</li> <li>4 S2 <b>and</b> P4 (double arrow) (<i>Parameters to Modify</i>)</li> <li>5 Return parameter from Delete ( )</li> </ol>	<b>5</b>

Question	Answer	Marks
3(b)	 <pre> graph TD     Design((Design)) --&gt; Coding((Coding))     Coding --&gt; Testing((Testing))     Testing --&gt; Design   </pre> <p>One mark for:</p> <ul style="list-style-type: none"> <li>• Design, Coding and Testing (in sequence)</li> <li>• Arrows as shown // Alternative 'waterfall model' where each stage loops back to the previous stage</li> </ul>	2

Question	Answer	Marks
4(a)	<pre> DECLARE Num : INTEGER Num ← 101  REPEAT   OUTPUT Num   Num ← Num + 2 UNTIL Num &gt; 199   </pre> <p>Mark as follows:</p> <ol style="list-style-type: none"> <li>1 Counter declaration and initialisation to sensible value <b>before the loop</b></li> <li>2 REPEAT ... UNTIL &lt;condition&gt;</li> <li>3 Correct selection of number to be output (use of MOD () or +2 or other)</li> <li>4 Correct range of numbers output</li> </ol>	4



Question	Answer	Marks
4(b)	<p>'Pseudocode' solution included here for development and clarification of mark scheme. Programming language example solutions appear in the Appendix.</p> <pre> FUNCTION Search(Par1, Par2 : STRING) RETURNS INTEGER  DECLARE Index, RetVal : INTEGER  Index ← 1 RetVal ← -1  WHILE Index &lt;= 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 </pre> <p>Mark as follows:</p> <ol style="list-style-type: none"> <li>Function heading and ending including parameters</li> <li>Declaration of local variable for array index (<i>Index</i>) but <b>not</b> of global <i>XRef</i> array</li> <li>Any loop for 100 elements</li> <li>Use of correct <i>XRef</i> 2D array syntax <b>in a loop</b></li> <li>Logical test of array elements <b>in a loop</b></li> <li>Exit loop if match found (following attempt at MP5) Return index number or -1 if not found</li> </ol>	7

Question	Answer	Marks
5(a)(i)	<p>One mark per point:</p> <ul style="list-style-type: none"> <li>Pretty print / Colour-coding of keywords / variables</li> <li>(Auto) indentation</li> <li>(Auto) Expansion / collapsing of data structures / code blocks // thumbnail overview</li> </ul> <p><b>Max 2 marks</b></p>	2
5(a)(ii)	<p>One mark per point:</p> <ul style="list-style-type: none"> <li>Dynamic syntax checking / highlighting syntax errors as code is typed</li> <li>Type checking</li> <li>Checking for used variables not declared / unused variables which are declared</li> </ul> <p><b>Max 2 marks</b></p>	2

Question	Answer	Marks
5(b)(i)	<p><b>DECLARE Response : BOOLEAN</b>            DECLARE Password : STRING            OUTPUT "Please Input your password: "            INPUT Password            Response ← Validate(UserID, Password) AND Today()  <b>RETURN Response</b></p> <p>One mark per line in <b>bold</b></p>	<b>2</b>
5(b)(ii)	<p>'Pseudocode' solution included here for development and clarification of mark scheme.            Programming language example solutions appear in the Appendix.</p> <pre> 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 &lt; 4     OUTPUT "Please input your password: "     INPUT Password     Response ← Validate(UserID, Password) AND TODAY()     IF Response = FALSE THEN       IF Count &lt; 3 THEN         OUTPUT "Incorrect password - please try again"       ENDIF       Count ← Count + 1     ENDIF   ENDWHILE    RETURN Response ENDFUNCTION           </pre> <p>1 mark for each of the following:</p> <ol style="list-style-type: none"> <li>1 If <u>parameter</u> matches "Guest" then <b>skip</b> to Return (MP7)</li> <li>2 Loop for max 3 attempts and terminate if valid password input</li> <li>3 Prompt and Input first password attempt</li> <li>4 Evaluate result of Validate() AND TODAY() <b>in a loop...</b></li> <li>5 Test result and increment count for incorrect password <b>in a loop...</b></li> <li>6 ...output 'try again' message and re-input password <b>in a loop...</b></li> <li>7 Return Boolean (correctly in all 3 cases)</li> </ol> <p><b>Note: Max 6 marks</b></p>	<b>6</b>

Question	Answer	Marks
6(a)	<p>'Pseudocode' solution included here for development and clarification of mark scheme. Programming language example solutions appear in the Appendix.</p> <pre> PROCEDURE StockSummary()   DECLARE Index, Count : INTEGER   DECLARE Total : REAL    Count ← 0   Total ← 0.0    FOR Index ← 1 TO 10000     IF StockID[Index] &lt;&gt; "" 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 </pre> <p>One mark for each of the following:</p> <ol style="list-style-type: none"> <li>1 Declaration <b>and</b> initialisation of Count and Total (by comment in Python)</li> <li>2 Loop for 10000 elements</li> <li>3 Skip row when StockID element = "" <b>in a loop</b></li> <li>4 Increment Count and sum Total <b>in a loop</b></li> <li>5 OUTPUT the total and the count <b>after the loop</b> (following a reasonable attempt at MP4)</li> </ol>	<b>5</b>

Question	Answer	Marks
6(b)	<pre> 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 &lt;= 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 </pre> <p>1 mark for each of the following:</p> <ol style="list-style-type: none"> <li>1 Function heading including input parameter <b>and</b> function End</li> <li>2 Declare local variable for line read from file</li> <li>3 OPEN file in READ mode <b>and</b> CLOSE</li> <li>4 Check whether file is empty <b>and</b> return FALSE if it is (no restore)</li> <li>5 Loop through all 10 000 elements initialising arrays</li> <li>6 Loop until EOF(Filename) OR Index &gt; 10000</li> <li>7 Call Unpack ( ) correctly for each line from the file <b>in a loop</b></li> <li>8 Return FALSE if more than 10000 lines in file, otherwise returns TRUE</li> </ol>	8

Question	Answer	Marks
6(c)	<p>'Pseudocode' solution included here for development and clarification of mark scheme. Programming language example solutions appear in the Appendix.</p> <pre> 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) &lt; 4 OR LENGTH(Filename) &gt; 10       THEN         Valid ← FALSE       ELSE         Index ← 1         WHILE Index &lt;= LENGTH(Filename) AND Valid = TRUE           NextChar ← MID(Filename, Index, 1)           IF NOT ((NextChar &gt;='a' AND NextChar &lt;='z')             OR (NextChar &gt;='A' AND NextChar &lt;='Z')             OR (NextChar &gt;='0' AND NextChar &lt;='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 </pre> <p>One mark for each of the following:</p> <ol style="list-style-type: none"> <li>1 Conditional loop until valid filename input</li> <li>2 Prompt and Input of filename <b>in a loop</b></li> <li>3 Test length is within range</li> <li>4 Loop through filename:</li> <li>5 Extract a single character</li> <li>6 Test that character is numeric <b>in a loop</b></li> <li>7 Test that character is alphabetic <b>in a loop</b></li> <li>8 If filename invalid, output warning and repeat, if valid then return filename</li> </ol>	<b>8</b>

\*\*\* 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;
    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;  
        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
```