Cambridge Assessment International Education

# Cambridge International AS & A Level

**COMPUTER SCIENCE** **9608/22**

Paper 2 Fundamental Problem-Solving and Programming Skills **October/November 2021**

MARK SCHEME

Maximum Mark: 75

**Published**

This document consists of **19** printed pages.

**[Turn over**

## 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.

| Question | Answer | Marks |
|---|---|---|
| 1(a) | One mark per bullet point to Max 4 e.g.<br>Design stage activities:<br>• create / produce / define identifier table // decide on identifiers<br>• create / produce / define data structures // data types<br>• create / produce / define file structures<br>• create / produce / define test plan / strategy<br>• create pseudocode<br>• create flowchart etc.<br>• identify inputs/outputs/processes<br>• decomposing the problem into sub-problems<br>• choose a suitable programming language<br><br>Coding stage activities:<br>• write program code // coding the algorithms (from design) 'write code' on its own is NE<br>• define data structures<br>• use a translator to check and run the code<br>• initial debugging // testing<br>• any example of actions performed when programming | **4** |
| 1(b)(i) | One mark per bullet point to Max 2<br>• names are not meaningful (or equivalent) // name does not reflect the identifier's use<br>• makes the program/variables more difficult to understand // difficult for non-technical/other person to understand the program/variables<br>• makes the program more difficult to debug / modify / test | **2** |
| 1(b)(ii) | One mark for each correct row.<br>**Line** – **Appropriate identifier name**<br>`102` – `Price / ItemPrice / StockPrice`<br>`103` – `ExpiryDate / ItemExpiryDate / EndDate`<br>`105` – `LowStockValue / LowValue / LowStock`<br>`106` – `IsOutOfStock / IsInStock` | **4** |
| 1(b)(iii) | One mark each<br>• (Constant) declaration with appropriate identifier for program version (as String)<br>• Storing correct value in the variable (equals or arrow)<br>e.g.<br>`CONSTANT ProgramVersion = "ver1.5.8"` | **2** |
| 1(c) | One mark each to Max 2<br>• wider range of character can be represented<br>• symbols from (more) languages can be represented<br>• pictograms / emoticons can be represented | **2** |

| Question | Answer | Marks |
|---|---|---|
| 2(a) | One mark each of the following to max 5:<br>• Input the membership number<br>• Open the file LOYALTY.txt for read **and** close the file<br>• Loop until end of file (and membership number not found)<br>• …read each line from the file<br>• Extract stored membership number **and** points<br>• If number of points is 10 or more, return/output free cake<br>• If membership number not found, return/output appropriate value/message<br>• If number of points is below 10, return / output no free cake<br>• Subtract 10 points from their points and store | **5** |
| 2(b) | One mark for each correct row. (see table below) | **4** |

| Description of expression | Pseudocode expression | Evaluates to |
|---|---|---|
| Evaluates to `TRUE` if `DayOFMonth` is within the first seven days of the month | `STRING_TO_NUM(DayOfMonth) <= 7`<br>`// DayOfMonth <= "7"` | `FALSE` |
| Concatenates the second and third letters of `Firstname` and concatenate with the last three letters of `Lastname` | `MID(Firstname,2,2) & RIGHT(Lastname, 3)`<br>`// MID(Firstname,2,2) & MID(Lastname, LENGTH(Lastname)-3, 3)` | `"eason"` |
| Evaluates to `TRUE` if `DOB` contains eight characters | `LENGTH(DOB) = 8` | `TRUE` |
| Evaluates to `TRUE` if the customer is a member and has enough points for a free slice of cake | `IsMember AND Points >= 10`<br>`IsMember = TRUE AND Points >= 10` | `TRUE` |

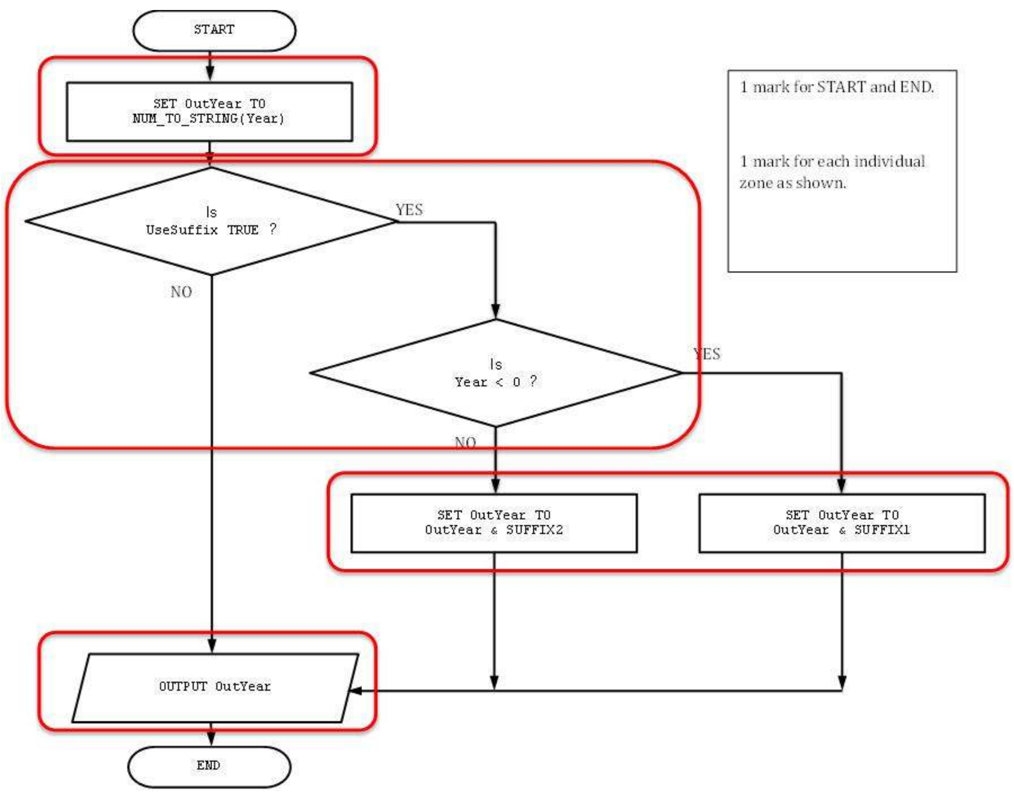
| Question | Answer | Marks |
|---|---|---|
| 2(c) | One mark for each bullet point<br><br>• CASE statement has correct structure (all key words and format)<br>• Correct assignment/identification of all 3 file types in case statement<br>• Assignment/return of `"Unknown"` as default<br>• Return `FileType` instead of `FileExt` // Correct return value for all file types and Unknown<br><br>`FUNCTION GetFileType(Filename : STRING) RETURNS STRING`<br>   `DECLARE FileExt : STRING`<br>   `DECLARE FileType: STRING`<br><br>   `FileExt ← RIGHT(Filename, 3)`<br><br>   `CASE OF FileExt`<br>      `'rtf' : FileType ← "Rich text format"`<br>      `'csv' : FileType ← "Comma separated values"`<br>      `'txt' : FileType ← "Text"`<br>      `OTHERWISE : FileType ← "Unknown"`<br>   `ENDCASE`<br><br>   `RETURN FileType`<br><br>`ENDFUNCTION` | 4 |

| Question | Answer | Marks |
|---|---|---|
| 3(a) | One mark each to max 3<br>• break the problem/algorithm (not program / code) into smaller steps / parts/ subproblems<br>• ... repeatedly only if MP1 given<br>• … until all subproblems small/detailed enough to solve<br>• … to identify program modules // to identify repeated elements // for modular programming<br>• … to identify subroutines | 3 |
| 3(b) | 1 mark for definition of term e.g.<br>• apply current knowledge to an unfamiliar scenario<br><br>1 mark for how the skills are used in program development e.g.<br>• use current knowledge of a familiar programming language in a new language | 2 |

| Question | Answer | Marks |
|---|---|---|
| 4(a) | One mark for each bullet point<br><br>• Procedure heading and ending<br>• Loop from 1 to 20 // Loop from 0 to 19 Allow NEXT for ENDFOR<br>• Assign −1 to array at loop counter<br>• All logic correct, does not override Flower array, all 20 elements assigned −1<br><br>e.g.<br>`PROCEDURE InitialiseArray()`<br><br>`  DECLARE Index : INTEGER`<br><br>`  FOR Index ← 1 TO 20`<br>`     Flower[Index] ← -1`<br>`  ENDFOR`<br><br>`ENDPROCEDURE` | **4** |

| Question | Answer | Marks |
|---|---|---|
| 4(b) | One mark for each bullet point:<br>• Declare variable to store first flower visited and initialise to 10 // Store flower 10 in first array index<br>• Loop until all 20 flowers are stored<br>• Generate random number between 1 and 20 **in the loop** …<br>• … Check if array at random number is −1 // Check if random number is already in flower array<br>• … If it is not −1 / already in array, generate another random number<br>• … repeatedly until −1 / not in array<br>• … otherwise assign Flower at previous flower to random number // otherwise assign random number to next array element and increment counter<br><br>'Pseudocode' solution included here for development and clarification of mark scheme. Programming language example solutions appear at the end.<br>Example 1:<br>`PROCEDURE RandomPath()`<br>`   DECLARE New : INTEGER`<br>`   DECLARE Previous : INTEGER`<br>`   New ← 10`<br>`   Previous ← 10`<br><br>`   FOR X ← 0 TO 19`<br>`      WHILE NOT(Flower[New] = -1)`<br>`        New ← INT(RAND(20)+1)`<br>`        Flower[Previous] ← New`<br>`      ENDWHILE`<br>`   ENDFOR`<br>`ENDPROCEDURE`<br>Example 2:<br>`PROCEDURE RandomPath()`<br><br>`   DECLARE Current : INTEGER`<br>`   DECLARE Counter: INTEGER`<br>`   DECLARE Next : INTEGER`<br><br>`   Current ← 10`<br>`   Counter ← 1`<br><br>`   WHILE Counter <= 20`<br>`      Next ← INT(RAND(20)+1)`<br><br>`      IF Flower[Next] = -1`<br>`        THEN`<br>`          Flower[Current] ← Next`<br>`          Current ← Next`<br>`          Counter ← Counter + 1`<br>`      ENDIF`<br><br>`   ENDWHILE`<br><br>`ENDPROCEDURE` | 7 |

| Question | Answer | Marks |
|---|---|---|
| 4(c)(i) | 1 mark for error type:<br>• Logic // run-time<br><br>1 mark for method of detection e.g.<br>• Testing the program with a range of data<br>• Whitebox testing<br>• Blackbox testing | **2** |
| 4(c)(ii) | 1 mark each to max 2 e.g.<br>• Program crashes<br>• Error as number is out of array bounds // number below 1 will produce error // number above 20 will produce error<br>• Not all elements will be accessed/used<br>• Error if decimal generated<br>• Error if negative number generated<br>• Infinite loop may be generated | **2** |

| Question | Answer | Marks |
|---|---|---|
| 5(a) | 1 mark each<br>• Start and end  Start/begin  End/stop Procedure/end procedure<br>• For each circled area (Max 4)<br> START<br>SET OutYear TO NUM_TO_STRING(Year)<br>Is UseSuffix TRUE ?<br>YES<br>NO<br>Is Year < 0 ?<br>YES<br>NO<br>SET OutYear TO OutYear & SUFFIX2<br>SET OutYear TO OutYear & SUFFIX1<br>OUTPUT OutYear<br>END<br>1 mark for START and END.<br>1 mark for each individual zone as shown. | **5** |

| Question | Answer | Marks |
|---|---|---|
| 5(b) | 1 mark for values, 1 mark for result from:<br>• `Year` >= 0 `UseSuffix` = TRUE<br>• Matching Year value with "CE"<br><br>• `Year` >= 0 `UseSuffix` = `FALSE`<br>• Matching Year value<br><br>• `Year` < 0 `UseSuffix` = `TRUE`<br>• Matching Year value with "BCE"<br><br>• `Year` < 0 `UseSuffix` = `FALSE`<br>• Matching Year value | **6** |
| 5(c)(i) | One mark each to max 2.<br>• code is unknown/not considered<br>• data is chosen to test boundaries // test with normal, extreme and erroneous<br>• compare expected output with actual output // testing whether the inputs produce the correct/expected outputs<br>• testing if the program meets the requirements | **2** |
| 5(c)(ii) | One mark each to max 2.<br>• data is chosen to test algorithm/code<br>• tests every path in the code<br>• test each line of code/structure<br>• internal structure is being tested<br>• tester can view code | **2** |

| Question | Answer | Marks |
|---|---|---|
| 6(a) | One mark each to max 6:<br><br>• Function declared taking the string to search for as a parameter, string return (and array)<br>• (Declare and) initialise a variable (0) to count number times code occurs<br>• Loop through 20 000 elements … // looping until unused element<br>• … IF Left 7 from GeoCodeLog[loop counter] = parameter …<br>• … if true, increment number times occurs<br>• … if true, RIGHT 10 from GeoCodeLog[loop counter] …<br>• … if true compare to last date and replace if after // store in variable for last date<br>• Return concatenated number times & " " & last date …<br>• … converting number to string<br>Example 1:<br>`FUNCTION SearchLog(SearchGeoCode : STRING) RETURNS STRING`<br>`   DECLARE AccessCount : INTEGER`<br>`   DECLARE LatestDate : STRING`<br>`   DECLARE DateAccess : DATE`<br><br>`   LatestDate ← "01/01/1500"`<br>`   FOR Index ← 0 TO 19999`<br>`      IF LEFT(GeoCodeLog[Index], 7)`<br>`         THEN`<br>`            AccessCount ← AccessCount + 1`<br>`            DateAccess ← (RIGHT(GeoCodeLog[Index],`<br>`                          10)).TODATE`<br>`            IF DateAccess > LatestDate`<br>`               THEN`<br>`                  LatestDate ← DateAccess`<br>`            ENDIF`<br>`      ENDIF`<br>`   ENDFOR`<br>`   RETURN NUM_TO_STRING(AccessCount) & " " &`<br>`          DATE_TO_STRING(LatestDate)`<br><br>`ENDFUNCTION` | **6** |

| Question | Answer | Marks |
|---|---|---|
| 6(a) | Example 2:<br>`FUNCTION SearchLog(SearchGeoCode : STRING) RETURNS STRING`<br><br>`   DECLARE Index : INTEGER`<br>`   DECLARE AccessCount : INTEGER`<br>`   DECLARE LogGeoCode : STRING`<br>`   DECLARE LastDate : STRING`<br>`   DECLARE CountDateLine : STRING`<br><br>`   CONSTANT SPACE = ' '`<br><br>`   Index ← 1`<br>`   AccessCount ← 0`<br>`   LogGeoCode ← ""`<br>`   LastDate ← ""`<br><br>`   WHILE Index <= 20000 AND LogGeoCode <> "AAAA+0A"`<br>`      LogGeoCode ← LEFT(GeoCodeLog[Index], 7)`<br>`      IF LogGeoCode = SearchGeoCode`<br>`        THEN`<br>`          AccessCount ← AccessCount + 1`<br>`          LastDate ← RIGHT(GeoCodeLog[Index], 10)`<br>`      ENDIF`<br>`      Index ← Index + 1`<br>`   ENDWHILE`<br><br>`   CountDateLine ← NUM_TO_STRING(AccessCount) & SPACE &`<br>`                   LastDate`<br><br>`   RETURN CountDateLine`<br><br>`ENDFUNCTION` | |

| Question | Answer | Marks |
|---|---|---|
| 6(b) | 1 mark each to max 7<br>• Procedure header<br>• Open file `"Locations.txt"` in WRITE mode **and** close the file<br>• Loop through 20 000 elements // looping until unused element …<br>• … if `GeoCodeData[loop counter] = "AAAA+0A"` loop again/exit<br>• … if not empty call `SearchLog()` with `LEFT(GeoCodeData[DataIndex], 7)` … FT invalid/missing if<br>• … use/store returned value<br>• … replace the space with a #<br>• … concatenate GeoCodeData[loop counter] & "#" & returned data **inside** the loop …<br>• … writing this value to file **inside** the loop<br><br>'Pseudocode' solution included here for development and clarification of mark scheme.<br>Programming language example solutions appear at the end.<br>Example 1:<br>`PROCEDURE ExtractArrays()`<br>`   DECLARE DataIndex : INTEGER`<br>`   DECLARE ShortCode, CountAndDate, LocationLine,`<br>`           TempCode : STRING`<br>`   CONSTANT FILENAME = "Locations.txt"`<br>`   DataIndex ← 1`<br>`   ShortCode ← ""`<br>`   TempCode ← ""`<br>`   OPENFILE FILENAME FOR WRITE`<br>`   FOR Count ← 0 TO 19999`<br>`     ShortCode ← LEFT(GeoCodeData[DataIndex], 7)`<br>`     IF NOT(ShortCode = "AAAA+0A") THEN`<br>`        CountAndDate ← SearchLog(ShortCode)`<br>`        FOR Index ← 0  TO LENGTH(CountAndDate)-1`<br>`           IF NOT(MID(CountAndDate,Index,1) = " ")`<br>`              THEN`<br>`                 TempCode = TempCode &`<br>`                            MID(CountAndDate,Index,1)`<br>`           ELSE`<br>`              TempCode = TempCode & "#"`<br>`           ENDIF`<br>`        ENDFOR`<br>`        LocationLine ← GeoCodeData[DataIndex] & "#" &`<br>`                        TempCode`<br>`        WRITEFILE(FILENAME, LocationLine)` | 7 |

| Question | Answer | Marks |
|---|---|---|
| 6(b) | <pre>      ENDIF<br>   ENDFOR<br>   CLOSEFILE FILENAME<br><br>ENDPROCEDURE</pre>Example 2:<pre>PROCEDURE ExtractArrays()<br><br>   DECLARE DataIndex : INTEGER<br>   DECLARE ShortCode, CountAndDate, LocationLine : STRING<br><br>   CONSTANT HASH = '#'<br>   CONSTANT FILENAME = "Locations.txt"<br><br>   DataIndex ← 1<br>   ShortCode ← ""<br><br>   OPENFILE FILENAME FOR WRITE<br><br>   WHILE DataIndex <= 20000 AND ShortCode <> "AAAA+0A"<br>      ShortCode ← LEFT(GeoCodeData[DataIndex], 7)<br>      IF ShortCode <> "AAAA+0A"<br>        THEN<br>          CountAndDate ← SearchLog(ShortCode)<br>          LocationLine ← GeoCodeData[DataIndex] & HASH &<br>                         CountAndDate<br>          WRITEFILE (FILENAME, LocationLine)<br>      ENDIF<br>      DataIndex ← DataIndex + 1<br>   ENDWHILE<br><br>   CLOSEFILE FILENAME<br><br>ENDPROCEDURE</pre> | |

**Program Code Example Solutions**

**Q4 (b): Visual Basic**

```
Sub RandomPath()
    ' alternative solution
    Dim Index, Current, NextFlower As Integer
    Dim AllFlowersVisited As Boolean
    Dim RandomFlower As New Random

    Current = 10
    AllFlowersVisited = False

    Do While AllFlowersVisited = False
        NextFlower = RandomFlower.Next(1, 20)

        If Flower(NextFlower) = 0 Then
            If Flower(Current) = 0 Then
                Flower(Current) = NextFlower
            End If
            Current = NextFlower
        End If
        AllFlowersVisited = True
        For Index = 1 To 20
            If Flower(Index) = 0 Then
                AllFlowersVisited = False
            End If
        Next
    Loop
End Sub
```

**Q4 (b): Pascal**

```
procedure RandomPath();
// alternative solution
var
   Index, Current, Next: integer;
   AllFlowersVisited : boolean;

begin
   Current := 10;
   AllFlowersVisited := False;

   while (AllFlowersVisited = False) do
   begin
      Randomize;
      Next := RandomRange(1, 21);

      if (Flower[Next] = 0) then
      begin
         if (Flower[Current] = 0) then Flower[Current] := Next;
         Current := Next;
      end;
      AllFlowersVisited := True;
      for Index := 1 to 20 do
      begin
         if Flower[Index] = 0 then
            AllFlowersVisited := False;
      end;
   end;
end;
```

**Q4 (b): Python**

```
from random import seed
from random import randint

def RandomPath():
    # alternative solution
    #DECLARE Index : INTEGER
    #DECLARE Current : INTEGER
    #DECLARE Next : INTEGER
    #DECLARE AllFlowersVisited : BOOLEAN

    Current = 10 #start at flower 10 in the field
    AllFlowersVisited = False

    while AllFlowersVisited == False:
        #loop until all flowers visited
        Next = randint(20)
        if Flower[Next] == 0:
            if Flower[Current] == 0: #do not revisit a flower
                Flower[Current] = Next
            Current = Next

        AllFlowersVisited = True #assume all visited unless...
        for Index in range(1, 20):
            if Flower[Index] == 0:
                AllFlowersVisited = False #... proved otherwise
```

**Q6 (b): Visual Basic**

```
Sub ExtractArrays()
   Dim DataIndex As Integer
   Dim ShortCode, CountAndDate, LocationLine, CountHashDate As String
   Dim Sr As StreamWriter = New StreamWriter("Locations")

  Const HASH = "#"

  DataIndex = 1
  ShortCode = ""
  CountHashDate = ""

  Do While DataIndex <= 20000 And ShortCode <> "AAAA+0A"
     CountHashDate = ""
     ShortCode = GeoCodeData(DataIndex).SubString(0, 7)
     If ShortCode <> "AAAA+0A" Then
       CountAndDate = SearchLog(ShortCode)
       For Index = 0 To CountAndDate.Length()-1
         If (MID(CountAndDate, Index, 1) = " ") Then
            CountHashDate = CountHashDate & "#"
         Else
           CountHashDate = CountHashDate & MID(CountAndDate, Index, 1) = ""
         End If
       Next Index
       LocationLine = GeoCodeData(DataIndex) & HASH & CountHashDate
       Sr.WriteLine(LocationLine)
     End If
     DataIndex = DataIndex + 1
  Loop
  Sr.Close()
End Sub
```

**Q6 (b): Pascal**

```
procedure ExtractArrays();
var
   DataIndex: integer;
   ShortCode, CountAndDate, LocationLine, CountHashDate: string;
   Locations: textfile;
const
   HASH = '#';
begin
   DataIndex := 1;
   ShortCode := '';

   assign(Locations,'M:\Locations');
   rewrite(Locations);

   while (DataIndex <= 20000) and (ShortCode <> 'AAAA+0A') do
   begin
      CountHashDate := "";
      ShortCode := LeftStr(GeoCodeData[DataIndex], 7);
      if ShortCode <> 'AAAA+0A' then
      begin
         CountAndDate := SearchLog(ShortCode);
        for Index := 0 to Length(CountAndDate)-1 do
            begin
              if CountAndDate[Index] = " " then
                 CountHashDate := CountHashDate + "#"
              else
                 CountHashDate := CountHashDate + CountAndDate[Index];
            end;


         LocationLine := GeoCodeData[DataIndex] + HASH + CountHashDate;
         writeln(Locations, LocationLine);
      end;
      DataIndex := DataIndex + 1;
   end;
   close(Locations);
end;
```

**Q6 (b): Python**

```
def ExtractArrays():
    #DECLARE DataIndex : INTEGER
    #DECLARE ShortCode, CountAndDate, LocationLine, CountHashDate : STRING

    HASH = "#"
    FILENAME = "Locations"

    DataIndex = 1
    ShortCode = ""

    filehandle = open(FILENAME, 'w')

    while dataIndex <= 20000 and ShortCode <> "AAAA+0A":
        ShortCode = GeoCodedata[DataIndex][:7]
       CountHashDate = ""
        if ShortCode <> "AAAA+0A":
            CountAndDate = SearchLog(ShortCode)
           for Index in range(0, Len(CountAndDate):
             if (CountAndDate[Index] == " "):
                 CountHashDate += "#"
             else:
                CountHashDate += CountAndDate[Index]
            LocationLine = GeoCodeData[DataIndex] + HASH + CountHashDate
            filehandle.write(LocationLine)
        DataIndex += 1

    filehandle.close()
```