Cambridge Assessment International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9608/21**

Paper 2 Written Paper **October/November 2020**

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 2020 series for most Cambridge IGCSE™, Cambridge International A and AS Level and Cambridge Pre-U components, and some Cambridge O Level components.

PUBLISHED

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, max 3 marks • Analysis / Problem Definition • Design • Code // Implement • Test	**3**
1(b)	One mark per bullet point • Corrective: changes to correct a bug / problem / error in the program • Adaptive: changes due to change in specification / requirements / legislation / available technology	**2**
1(c)	One mark per bullet point • The knowledge / experience / understanding of one programming language • Can be applied to another / unfamiliar language // will help recognise control structures in another / unfamiliar language // will help them learn a new language	**2**
1(d)	One mark per bullet point • Names are not meaningful (or equivalent) // name does not reflect the identifier's use// easy to use the wrong name • Makes the program more difficult to understand / debug / modify / test	**2**

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>1(e)</td><td>1 mark for any two rows correct, 2 marks for all rows correct.

<table>
<tr><th>Expression</th><th>Evaluates to</th></tr>
<tr><td><code>Alarm OR NOT PowerFail</code></td><td>TRUE</td></tr>
<tr><td><code>NOT (Alarm AND PowerFail)</code></td><td>FALSE</td></tr>
<tr><td><code>(GateOpen OR Alarm) AND PowerFail</code></td><td>TRUE</td></tr>
<tr><td><code>(GateOpen AND Alarm) OR NOT PowerFail</code></td><td>FALSE</td></tr>
</table></td><td>2</td></tr>
<tr><td>2(a)</td><td>One mark per step (or equivalent) to max 6 marks

1 Prompt / output string and input the new user name
2 OPEN the file in read mode and close the file
3 Initialise a Boolean variable, e.g. <code>UniqueFlag to TRUE</code>
4 LOOP while not End of File (AND <code>new user name not found)</code>
5 Read a line from the file
6 If line is same as new user name (then set termination condition else repeat from step 4)
7 IF <code>UniqueFlag</code> is TRUE then output ‘Unique’ otherwise output ‘Not unique’</td><td>6</td></tr>
</table>

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>2(b)</td><td>One mark per row (for expression completed by addition of text shown in bold):

<table>
<tr><th>Expression</th><th>Evaluates to</th></tr>
<tr><td>LEFT("Stepwise", 2) & "art"</td><td>"Start"</td></tr>
<tr><td>MID("Concatenate", 6, 3)</td><td>"ten"</td></tr>
<tr><td>2 * LENGTH("Kipper")</td><td>12</td></tr>
<tr><td>TRUE OR FALSE</td><td>TRUE</td></tr>
<tr><td>MOD(9, 2)</td><td>1</td></tr>
</table></td><td>5</td></tr>
<tr><td>2(c)</td><td>One mark per row:

<table>
<tr><th></th><th>Answer</th></tr>
<tr><td>The name for the type of loop used</td><td>Count-controlled</td></tr>
<tr><td>A line number of a selection statement</td><td>14</td></tr>
<tr><td>The scope of OutString</td><td>Local</td></tr>
<tr><td>The name of a function that is called</td><td>LENGTH // MID // LCASE</td></tr>
<tr><td>A line number containing a logical operator</td><td>14</td></tr>
</table></td><td>5</td></tr>
</table>

Question	Answer	Marks
3(a)	Reasons include: 1 `FileName` given does not exist / access denied / file is of wrong type 2 The `StartLine` line does not exist in file 3 There are less than `NumLines` lines after the `StartLine` 4 Code does not read the required number of lines (description of logical error) One mark per point to 3 max marks	**3**

3(b)	'Pseudocode' solution included here for development and clarification of mark scheme.
	Programming language example solutions appear in the Appendix.
	
	`PROCEDURE OutputLines(FileName: STRING, StartLine, NumLines: INTEGER)`
	
	`  DECLARE FileData : STRING`
	`  DECLARE Count : INTEGER`
	
	`  OPENFILE FileName FOR READ`
	
	`  FOR Count ← 1 TO StartLine + NumLines - 1`
	`  READFILE FileName, FileData`
	`  IF Count >= StartLine`
	`  THEN`
	`  OUTPUT FileData`
	`  ENDIF`
	`  ENDFOR`
	
	`  CLOSE FileName`
	
	`ENDPROCEDURE`
		7

Question	Answer	Marks

3(b)	1 mark for each of the following:
	
	1 Procedure heading (including parameters) and ending
	2 Declare local variable for `FileData` and `Count`
	3 Open `FileName` in `READ` mode **and** subsequent Close
	4 Loop
	5 Read `Filedata` **in a loop**
	6 Check if current line is to be printed (i.e. if `Count` in range)
	7 If so Output `FileData` **in a loop**
	
	Alternative 'Seek' solution
	
	`PROCEDURE OutputLines(FileName: STRING, StartLine, NumLines: INTEGER)`
	
	`DECLARE FileData : STRING`
	`DECLARE Count : INTEGER`
	
	`OPENFILE FileName FOR READ`
	
	`FOR Count ← 1 TO StartLine - 1 // read up to StartLine`
	`READFILE FileName, FileData`
	`ENDFOR`
	
	`FOR Count ← 1 TO NumLines // then Output NumLines lines from here`
	`READFILE FileName, FileData`
	`OUTPUT FileData`
	`ENDFOR`
	
	`CLOSE FileName`
	
	`ENDPROCEDURE`	

Question	Answer	Marks
3(b)	1 mark for each of the following: 1 Procedure heading (including parameters) and ending 2 Declare local variable for `FileData` and `Count` 3 Open `FileName` in `READ` mode **and** subsequent close 4 **First** loop to read up to `StartLine` 5 Read `Filedata` **in a loop** 6 **Second** loop for `Numlines` times 7 Read **and** Output `Filedata` **in a loop**	
3(c)(i)	One mark for type plus one for corresponding explanation to max 2 marks • Logic(al) Error • The program does not perform as expected (or by example) **OR** • Run-time error • The program executes an invalid instruction / attempts to divide by zero // the program crashes Award mark for 'description' without type	**2**
3(c)(ii)	Max 2 marks, techniques include: • White-box testing // use of suitable test data • Dry-run / use of a trace table • IDE features such as breakpoints, stepping, watch windows	**2**
3(d)	Max 2 marks, reasons include: • They are tried and tested / free from bugs • They are already available so their use saves development time • Perform a function which the programmer does not have the skills to write	**2**

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>4(a)</td><td>There are many correct answers.

There are eight relevant combinations (2 × 2 × 2 options) as follows:

<table>
<tr><th>Parameter</th><th>Option A</th><th>Option B</th></tr>
<tr><td><code>Number</code></td><td><= 999</td><td>>= 1000</td></tr>
<tr><td><code>Prefix</code></td><td>Empty String</td><td>Non-empty String</td></tr>
<tr><td><code>AddComma</code></td><td>TRUE</td><td>FALSE</td></tr>
</table>

Tests must be different by at least one option

For each test (2 in total):

• One mark for parameter values
• One mark for expected return</td><td>4</td></tr>
</table>

Question	Answer	Marks

4(b)	`FUNCTION FormOut(Num : INTEGER, Prefix : STRING, AddComma : BOOLEAN) RETURNS STRING`
	
	`  DECLARE OutString : STRING`
	`  DECLARE ThouDigits : INTEGER    // just to simplify later expression`
	
	`  CONSTANT COMMA = ','`
	
	`  OutString ← NUM_TO_STRING(Num) // convert integer to string`
	`  IF AddComma = TRUE`
	`  THEN`
	`  NumLength ← LENGTH(OutString)`
	`  IF NumLength > 3`
	`  THEN`
	`  ThouDigits ← NumLength - 3`
	`  OutString ← LEFT(OutString, ThouDigits) & COMMA & RIGHT(OutString, 3)`
	`  ENDIF`
	`  ENDIF`
	
	`  OutString ← Prefix & OutString //  concatenate string with Prefix`
	
	`  RETURN OutString`
	
	`ENDFUNCTION`
		8

Question	Answer	Marks
4(b)	1 mark for each of the following: 1 Functon heading and ending (inc parameters) and return data type 2 Use of `NUM_TO_STRING` 3 Test `AddComma` and if `TRUE` … 4 Test number of digits in `Num` 5 Split `NUM_TO_STRING(Num)` and 6 Form `OutString` with inserted comma 7 Concatenate the `Prefix` with `OutString` 8 Return `OutString` MP3 and MP4 could easily appear in reverse sequence	

Question	Answer	Marks

5(a)	`FUNCTION GetIndex(HashTag : STRING) RETURNS INTEGER`
	
	`  DECLARE Index : INTEGER`
	`  DECLARE Found : BOOLEAN`
	
	`  Index ← 1`
	`  Found ← FALSE`
	
	`  WHILE Index <= 10000 AND Found = FALSE`
	`  IF TagString[Index] = HashTag`
	`  THEN`
	`  Found ← TRUE`
	`  ELSE`
	`  Index ← Index + 1`
	`  ENDIF`
	`  ENDWHILE`
	
	`  IF Found = FALSE`
	`  THEN`
	`  Index ← -1`
	`  ENDIF`
	
	`  RETURN Index`
	`ENDFUNCTION`
	
	1 mark for each of the following:
	
	1 Declaration of `Index` as Integer and `Found` as Boolean // `RetIndex` as Integer if this used for loop termination
	2 Initialisation of `Index`
	3 Conditional loop for 10 000 elements while `HashTag` not found
	4 Compare `HashTag` with element from `TagString` array **in a loop**
	5 Set termination condition / store current index if match found **in a loop**
	6 Return Integer value
		6

Question	Answer	Marks

5(b)	`FUNCTION GetStart(Message : STRING, TagNum : INTEGER) RETURNS INTEGER`
	
	`DECLARE MessLength, Index, StartPos, Count: INTEGER`
	`CONSTANT HASH = '#'`
	
	`Count ← 0`
	`StartPos ← -1`
	`Index ← 1`
	`MessLength ← LENGTH(Message)`
	
	`WHILE Index <= MessLength AND Count < TagNum`
	`IF MID(Message, Index, 1) = HASH`
	`THEN`
	`Count ← Count + 1`
	`IF Count = TagNum`
	`THEN`
	`StartPos ← Index // found the required hashtag`
	`ENDIF`
	`ENDIF`
	`Index ← Index + 1`
	`ENDWHILE`
	
	`RETURN StartPos`
	`ENDFUNCTION`
	
	1 mark for each of the following:
	
	1 Conditional loop until required hashtag found or end of message encountered
	2 Extract a character **and** compare with hash character **in a loop**
	3 If hash found, increment `Count / decrement TagNum`
	4 Test if this is the required hashtag
	5 Set termination condition / store current index if match found **in a loop**
	6 Return integer value from correct function declaration
		6

Question	Answer	Marks
5(c)	'Pseudocode' solution included here for development and clarification of mark scheme. Programming language example solutions appear in the Appendix. Max 8 marks from 9 mark points	**8**

```
FUNCTION GetTag(Message : STRING, StartPos : INTEGER) RETURNS STRING
   DECLARE Index : INTEGER
   DECLARE MyString : STRING
   DECLARE NextChar, TestChar : CHAR
   DECLARE EndTag : BOOLEAN
   CONSTANT HASH = '#'
   CONSTANT SPACE = ' '

   MyString ← ""
   EndTag ←FALSE

   IF MID(Message, StartPos, 1) = HASH // check first char is HASH
     THEN
        MyString ← HASH               // store HASH as first char of hashtag
        Index ← StartPos + 1          // start loop with first char after #

        WHILE Index <= LENGTH(Message) AND EndTag = FALSE
           NextChar ← MID(Message, Index, 1)
           TestChar ← UCASE(NextChar)
           IF TestChar = SPACE OR TestChar = HASH __
                               OR NOT ((TestChar >= 'A' AND TestChar <= 'Z') __
                               OR (TestChar >= '0' AND TestChar <= '9'))
              THEN
                 EndTag ← TRUE
              ELSE
                 MyString ← MyString & NextChar
           ENDIF
           Index ← Index + 1
        ENDWHILE
```

Question	Answer	Marks

5(c)	```
	   ENDIF
	   IF LENGTH(MyString) = 1
	      THEN
	         MyString = "" // when Hash is last char in string
	   ENDIF
	   RETURN MyString
	ENDFUNCTION
	``
	
	1 mark for each of the following:
	
	1 Function heading and ending including parameters and return data type (not python)
	2 Declaration and initialisation of return string (`MyString`)
	3 Test for valid start character
	4 Loop from `StartPos+1` while not end of message
	5 Extract `NextChar`
	6 Test if `NextChar` is terminator character **in a loop**
	7 If `NextChar` not a terminator then concatenate with `MyString` **in a loop**
	8 Cater for `MyString` only containing a Hash character (return empty string)
	9 Return `MyString`	

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

Program Code Example Solutions

Q3 (b): Visual Basic

```
Sub OutputLines(FileName As String, StartLine As Integer, NumLines As Integer)

  Dim FileData As String
  Dim Count As Integer
  Dim File As New StreamReader(FileName)

  For Count = 1 To StartLine + NumLines - 1
     FileData = File.ReadLine()
     If Count >= StartLine Then Console.Writeline(FileData)
  Next

File.Close

End Sub
```

Q3 (b): Pascal

```
Procedure OutputLines(FileName : string; StartLine : integer; NumLines : integer);

var
  FileData : string;
  Count : integer;
  ThisFile : textfile;

begin
  assignfile(ThisFile, FileName);
  reset(ThisFile);

  for Count := 1 to StartLine + NumLines - 1 do
     begin
        readln(ThisFile, Filedata);
        if Count >= StartLine then writeln(Filedata);
     end;

closefile(ThisFile);
end;
```

Q3 (b): Python

```
def OutputLines(FileName, StartLine, NumLines):
  #Count as INTEGER
  #File as FILEOBJECT
  #FileData as STRING

  Count = 1
  File = open(FileName, "r")

  for Count in range(1, StartLine + NumLines):
     FileData = File.readline()
     if Count >= StartLine:
        print(FileData)

  File.close()
```

Q5 (c): Visual Basic

```
Function GetTag(Message As String, StartPos As Integer) As String

  Dim NumChars, Index As Integer
  Dim MyString As String
  Dim NextChar, TestChar As Char
  Dim EndTag As Boolean
  Const HASH = "#"
  Const SPACE = " "

  MyString = ""
  EndTag = FALSE

  NumChars = Len(Message)

  If Mid(Message, StartPos, 1) = HASH Then   'Check for valid hashtag
     MyString = HASH
     Index = StartPos + 1
     Do While Index <= NumChars And EndTag = FALSE
        NextChar = Mid(Message, Index, 1)
        TestChar = Ucase(NextChar)
        If NextChar = SPACE Or NextChar = HASH __
           Or Not ((TestChar >= 'A' And TestChar <= 'Z') Or __
                   (TestChar >= '0' And TestChar <= '9')) Then
           EndTag = TRUE
        Else
           MyString = MyString & NextChar
        End If
        Index = Index + 1
     Loop
  End If

  If MyString = HASH Then MyString = ""  'When Hash is last char in message

  Return MyString

End Function
```

Q5 (c): Pascal

```
Function GetTag(Message : string, StartPos : integer) : String;
  var
   NumChars, Index : Integer;
   MyString: String;
   NextChar, TestChar : String;
   EndTag : Boolean;

  const
   HASH = '#';
   SPACE = ' ';

  MyString = '';
  EndTag = FALSE;


  NumChars := Length(Message);

  if midstr(Message, StartPos, 1) = HASH then // Check for valid hashtag
  begin
     MyString:= HASH;
     Index := StartPos + 1;
     while Index <= NumChars And EndTag = FALSE do;
     begin
        NextChar := Midstr(Message, Index, 1);
        TestChar := UpperCase(NextChar);
        If NextChar = SPACE OR NextChar = HASH Or
           Not ((TestChar >= 'A' And TestChar <= 'Z') Or
                 TestChar >= '0' And TestChar <= '9')) then
           EndTag := TRUE;
        Else
           MyString:= MyString + NextChar;
        Index := Index + 1;
     end;
  end;
```

```
   If MyString = HASH then MyString := ""; // When Hash is last char in message

  GetTag := MyString;

end;
```

Q5 (c): Python

```
def GetTag(Message, StartPostion):

#EndTag as BOOLEAN
#Index as INTEGER
#MyString as STRING
#NextChar, TestChar as CHAR

  HASH = '#'
  SPACE = ' '

  MyString = ""
  EndTag = FALSE


  NumChars = len(Message)

  if message[Index] == "#":
     MyString = HASH
     Index = StartPos + 1

     while Index <= NumChars And EndTag == FALSE:
        NextChar = Message[Index]
        TestChar = NextChar.upper()
        if NextChar == SPACE OR NextChar == HASH
           or not ((TestChar >= 'A' and TestChar <= 'Z') or
                    TestChar >= '0' and TestChar <= '9')):
           EndTag = True
        else:
           MyString = MyString + NextChar
        Index = Index + 1
```

```
  if MyString = HASH:
     MyString = ""  #When Hash is last char in message

  return HashTag
```