
COMPUTER SCIENCE**9608/22**

Paper 2 Written Paper

October/November 2019

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 2019 series for most Cambridge IGCSE™, Cambridge International A and AS Level components and some Cambridge O Level components.

This document consists of **13** 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.

| Question | Answer | Marks |
|----------|---|----------|
| 1(a)(i) | Any three from: <ol style="list-style-type: none"> 1. Indentation 2. Blank lines / white space 3. Sensible identifier names / use of Camel Case for identifier names 4. Capitalised keywords | 3 |
| 1(a)(ii) | <div style="display: flex; justify-content: space-between;"> <div style="width: 45%;"> <pre> graph TD Start([START]) --> Tries[Tries ← 1] Tries --> Full[Full ← ReadSensor("F1")] Full --> IsFull{Is Full = TRUE?} IsFull -- YES --> OutputFull[/Output "Already Full"/] IsFull -- NO --> IsNotFull{Is NOT Full AND Tries < 4} IsNotFull -- YES --> TopUp[Call TopUp()] TopUp --> ReadSensor[ReadSensor("F1")] ReadSensor --> TriesInc[Tries ← Tries + 1] TriesInc --> IsNotFull IsNotFull -- NO --> IsTries{Is Tries > 3?} IsTries -- YES --> OutputAttempts[/Output "Too many attempts"/] IsTries -- NO --> OutputFullTank[/Output "Tank now full"/] OutputFullTank --> End([END]) OutputAttempts --> End OutputFull --> End </pre> </div> <div style="width: 45%;"> <p>Mark as follows:</p> <ul style="list-style-type: none"> • One mark for START and END • One mark for each of the 4 areas outlined <p>Loop must be included for this area mark</p> </div> </div> | 5 |

| Question | Answer | Marks | | | | | | | | | | |
|----------------------------------|--|---------------|--------------|----------------------------------|---------|--------------------------------|---------|-----------------------------|-------|-----------|--------|---|
| 1(b)(i) | <p>One mark per row</p> <table border="1"> <thead> <tr> <th>Example value</th> <th>Data type</th> </tr> </thead> <tbody> <tr> <td>43</td> <td>INTEGER</td> </tr> <tr> <td>TRUE</td> <td>BOOLEAN</td> </tr> <tr> <td>-273.16</td> <td>REAL</td> </tr> <tr> <td>"-273.16"</td> <td>STRING</td> </tr> </tbody> </table> | Example value | Data type | 43 | INTEGER | TRUE | BOOLEAN | -273.16 | REAL | "-273.16" | STRING | 4 |
| Example value | Data type | | | | | | | | | | | |
| 43 | INTEGER | | | | | | | | | | | |
| TRUE | BOOLEAN | | | | | | | | | | | |
| -273.16 | REAL | | | | | | | | | | | |
| "-273.16" | STRING | | | | | | | | | | | |
| 1(b)(ii) | <p>One mark per row</p> <table border="1"> <thead> <tr> <th>Expression</th> <th>Evaluates to</th> </tr> </thead> <tbody> <tr> <td>RIGHT("Stop", 3) & LEFT("ich",2)</td> <td>"topic"</td> </tr> <tr> <td>MID(NUM_TO_STRING(2019), 3, 1)</td> <td>"1"</td> </tr> <tr> <td>INT(NUM_TO_STRING(-273.16))</td> <td>ERROR</td> </tr> <tr> <td>INT(13/2)</td> <td>6</td> </tr> </tbody> </table> | Expression | Evaluates to | RIGHT("Stop", 3) & LEFT("ich",2) | "topic" | MID(NUM_TO_STRING(2019), 3, 1) | "1" | INT(NUM_TO_STRING(-273.16)) | ERROR | INT(13/2) | 6 | 4 |
| Expression | Evaluates to | | | | | | | | | | | |
| RIGHT("Stop", 3) & LEFT("ich",2) | "topic" | | | | | | | | | | | |
| MID(NUM_TO_STRING(2019), 3, 1) | "1" | | | | | | | | | | | |
| INT(NUM_TO_STRING(-273.16)) | ERROR | | | | | | | | | | | |
| INT(13/2) | 6 | | | | | | | | | | | |

| Question | Answer | Marks |
|----------|---|-------|
| 2(a) | <p>One mark per point:</p> <ol style="list-style-type: none"> The source code represents a solution / design / algorithm expressed in a high-level language The Object code is produced (by the compiler) during the translation stage // The Object code is produced by translating the source code (NOT produced by Interpreter) Corrective maintenance occurs when testing reveals a fault (or error) in the program and this is corrected // Corrective maintenance is when errors are found and fixed // Corrective maintenance is when a program is debugged <p>Accept alternative answers provided they relate to the program development cycle</p> | 3 |
| 2(b) | <p>Any three from:</p> <ol style="list-style-type: none"> Dynamic syntax checking // Identification of syntax errors Highlighting undeclared variables // incorrect variable usage... Parameter checking Type checking Auto-indentation PrettyPrint | 3 |

| Question | Answer | Marks |
|----------|---|----------|
| 3(a) | <pre> DECLARE Result : ARRAY [0:9] OF INTEGER DECLARE Index : INTEGER FOR Index ← 0 TO 9 Result[Index] ← 0 ENDFOR </pre> <p>One mark for each of the following:</p> <ol style="list-style-type: none"> 1. Declaration of RESULT array (10 elements of type INTEGER) 2. Loop 3. Assignment within a loop | 3 |
| 3(b) | <pre> DECLARE Index : INTEGER DECLARE NextChar : CHAR DECLARE NextCharValue : INTEGER FOR Index ← 1 TO LENGTH(InString) NextChar ← MID(InString, Index, 1) NextCharValue ← STRING_TO_NUM(NextChar) Result[NextCharValue] ← Result[NextCharValue] + 1 ENDFOR FOR Index ← 0 TO 9 OUTPUT "Count of digit " & NUM_TO_STRING(Index) & " : " & NUM_TO_STRING(Result[Index]) ENDFOR </pre> <p>One mark for each of the following:</p> <ol style="list-style-type: none"> 1. Declaration of INTEGER variable for Index (or equivalent) to index Result array 2. First loop from 1 to length of InString: 3. Select each character (e.g MID) in first loop 4. Apply type conversion to obtain integer value for index in first loop 5. Increment element of Result array in a loop 6. Separate second loop to repeat 10 times: 7. Attempt to OUTPUT two items (digit 0 to 9 plus corresponding count) in any loop 8. OUTPUT statement including index and count in any loop including type conversion of element from array if required in a loop | 8 |

| Question | Answer | Marks |
|----------|---|-------|
| 4(a) | <pre> 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 ENDIF ENDPROCEDURE </pre> <p>Mark as follows:</p> <ol style="list-style-type: none"> 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 <p>If attempting to increase volume (parameter value was 10):</p> <ol style="list-style-type: none"> 5. Increment VolLevel 6. Ensure VolLevel still in range: for both cases. i.e: VolLevel ≤ 49 (for MaxVol = 0) VolLevel ≤ MaxVol (for MaxVol > 0) | 6 |
| 4(b) | <p>2 independent marks for each test:</p> <p>Test 1 MaxVol: 0/49 VolLevel expected: 49</p> <p>Test 2 VolLevel before: 34 VolLevel expected: 34</p> <p>Test 3 Parameter: 20 VolLevel expected: 0</p> | 6 |

| Question | Answer | Marks |
|----------|---|-------|
| 4(c)(i) | One mark for type, one for description: <ul style="list-style-type: none"> 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 | 2 |
| 4(c)(ii) | One mark per bullet point: <ul style="list-style-type: none"> 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 |

| Question | Answer | Marks |
|----------|---|-------|
| 5 | <div style="text-align: center;"> <pre> graph TD Renew[Renew()] Validate[Validate()] Lookup[Lookup()] Update[Update()] Validate -- P3 --> Renew Renew -- S2 --> Validate Lookup -- P4 --> Renew Renew -- M4 --> Lookup Update -- T4 --> Renew </pre> </div> <p>One mark for each of:</p> <ol style="list-style-type: none"> Diagram with boxes as above correctly labelled P3 and S2 Return Boolean from <code>Validate()</code> P4 M4 T4 Return parameter from <code>Update()</code> | 7 |

| 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> FUNCTION SearchLeavers(Reference : STRING) RETURNS BOOLEAN DECLARE Index : INTEGER DECLARE Found : BOOLEAN Found ← FALSE Index ← 0 WHILE Index < 500 AND NOT Found IF Reference = Leavers[Index] THEN Found ← TRUE ENDIF Index ← Index + 1 ENDWHILE RETURN Found ENDFUNCTION </pre> <p>One mark for each of the following:</p> <ol style="list-style-type: none"> Function heading (and ending) as above Initialisation and increment of <code>Index</code> used to index <code>Leavers</code> array in a loop Conditional loop repeating while <code>Index < 500</code> and exit loop if <code>Reference</code> is found: Compare indexed array element value with <code>Reference</code> in a loop Set termination logic if found in a loop Return Boolean value | 6 |

| Question | Answer | Marks |
|----------|--|-------|
| 6(b) | <pre> FUNCTION ProcessStudentList() RETURNS INTEGER DECLARE NotCopied : INTEGER DECLARE FileData : STRING DECLARE Reference : STRING NotCopied ← 0 OPENFILE "StudentList.txt" FOR READ OPENFILE "UpdatedList.txt" FOR WRITE WHILE NOT EOF("StudentList.txt") READFILE "StudentList.txt", FileData IF MID(FileData, 6, 1) = '*' THEN Reference ← MID(FileData, 1, 5) // five char reference ELSE Reference ← MID(FileData, 1, 8) // eight char reference ENDIF IF SearchLeavers(Reference) = FALSE THEN WriteFile "UpdatedList.txt", FileData ELSE NotCopied ← NotCopied + 1 ENDIF ENDWHILE CLOSEFILE "StudentList.txt" CLOSEFILE "UpdatedList.txt" RETURN NotCopied ENDFUNCTION </pre> <p>One mark for each of the following:</p> <ol style="list-style-type: none"> 1. Function heading and ending as above 2. Declaration and use of three local variables and initialisation of count to 0 3. OPEN both files in correct mode and CLOSE 4. Pre-Condition loop to go through the file StudentList.txt until EOF() 5. Read line from StudentList.txt and extract correct Reference (either 5 or 8 characters) in a loop 6. Call SearchLeavers() with Reference (after attempted extraction) in a loop 7. If result is FALSE then write FileData to UpdatedList.txt in a loop 8. Otherwise increment NotCopied in a loop 9. Return NotCopied count | 9 |

| 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> <p>One mark per underlined section:</p> <pre><u>Result</u> ← <u>CountTimes</u> (<u>Leavers</u>, "<u>"</u>)</pre> <p>(Space before open bracket to highlight underlined section only)</p> | 3 |

Program Code Example Solutions

Q6 (a): Visual Basic

```
Function SearchLeavers(ByVal Reference As String) As Boolean

    Dim Index As Integer
    Dim Found As Boolean

    Found = FALSE
    Index = 0

    Do While Index < 500 And Not Found

        If Reference = Leavers(Index) Then
            Found = TRUE
        End If

        Index = Index + 1

    Loop

    Return Found

End Function
```

Q6 (a): Pascal

```
function SearchLeavers(Reference : String) : boolean;

  var Index : integer;
  var Found : boolean;

  begin
    Found := FALSE;
    Index := 0;

    While Index < 500 And Not Found
    begin
      if Reference = Leavers[Index] Then
        Found := TRUE;

        Index := Index + 1;

      end;

    result := Found; // SearchLeavers := Found

  end;
```

Q6 (a): Python

```
def SearchLeavers(Reference):

    ## Index : Integer
    ## Found : Boolean

    Found = False
    Index = 0

    while Index < 500 and not Found:
        if Reference == Leavers[Index]:
            Found = True

            Index = Index + 1

    return Found
```

Q6 (c): Visual Basic

```
Result = CountTimes(Leavers, "")
```

Q6 (c): Pascal

```
Result := CountTimes(Leavers, "");
```

Q6 (c): Python

```
Result = CountTimes(Leavers, "")
```