Cambridge International AS & A Level

**Cambridge Assessment International Education**
Cambridge International Advanced Subsidiary and Advanced Level

**COMPUTER SCIENCE** **9608/23**

Paper 2 Written Paper **May/June 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 May/June 2019 series for most Cambridge IGCSE™, Cambridge International A and AS Level and Cambridge Pre-U components, and some Cambridge O Level components.



© UCLES 2019

## 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.

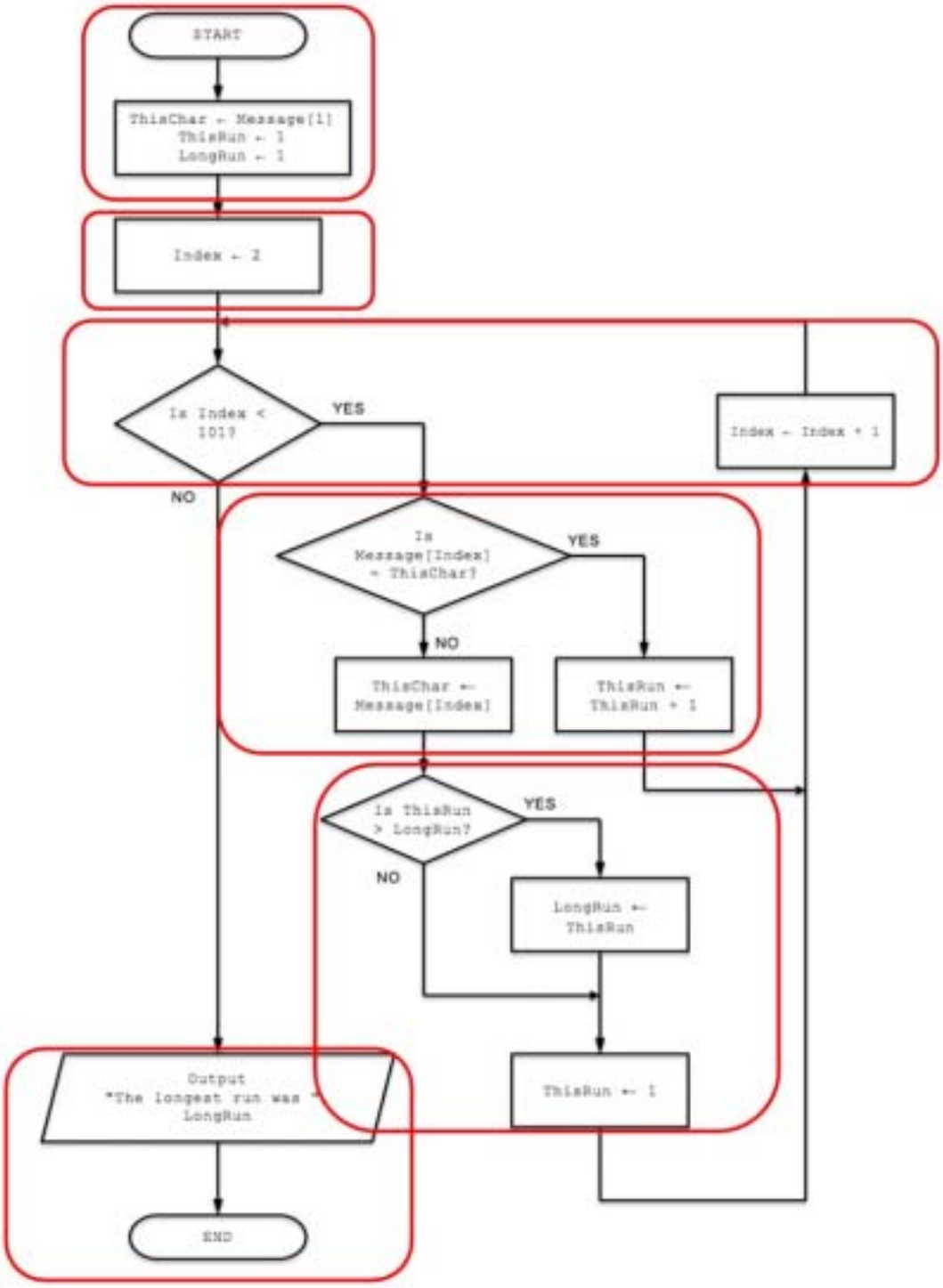
GENERIC MARKING PRINCIPLE 5:

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 2019

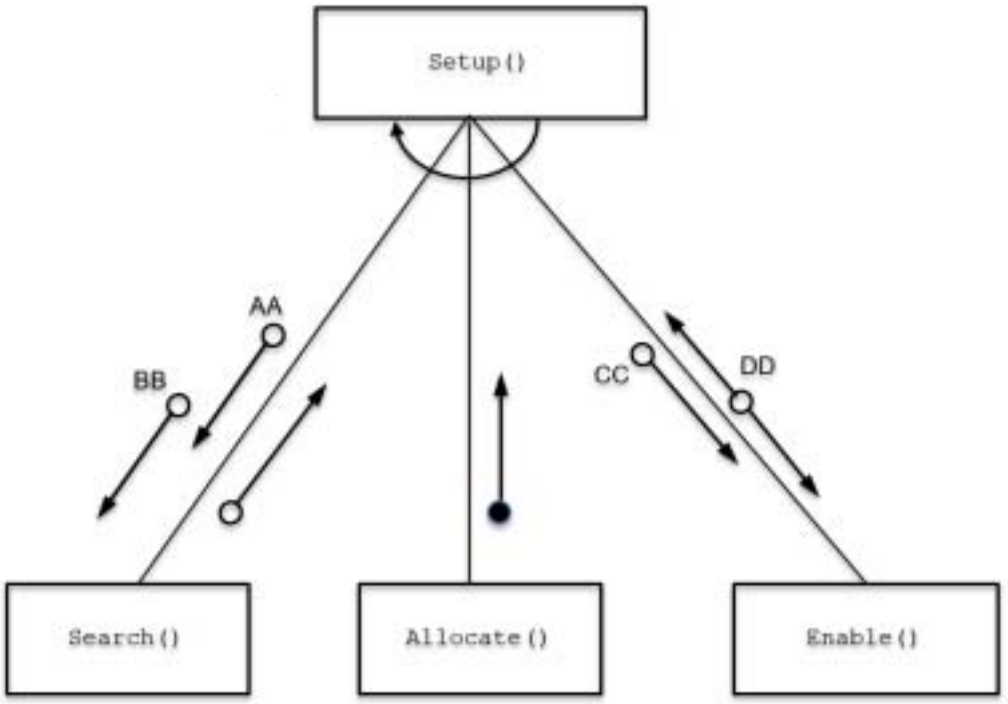
| Question | Answer | Marks |
|---|---|---|
| 1(a) | <br>Mark as follows:<br>• One mark per area outlined, in correct place<br>• Decision must be diamond symbol and have two outputs with at least one label (YES / NO) | **6** |

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>1(b)(i)</td><td>
<table>
<tr><th>Expression</th><th>Evaluates to</th></tr>
<tr><td>STRING_TO_NUM(MID(ProductID, 3, 2)) + 4</td><td>31.0 / 31</td></tr>
<tr><td>INT(MeltingPoint / 2)</td><td>90</td></tr>
<tr><td>Soluble AND Attempt > 3</td><td>FALSE</td></tr>
<tr><td>LENGTH(ProductID & NUM_TO_STRING(MeltingPoint))</td><td>11</td></tr>
<tr><td>RIGHT(ProductName, 4) & MID(ProductName, 5, 4)</td><td>"postroom"</td></tr>
</table>
Quotes for row 5 only
</td><td>5</td></tr>
<tr><td>1(b)(ii)</td><td>
<table>
<tr><th>Variable</th><th>Data type</th></tr>
<tr><td>MeltingPoint</td><td>REAL</td></tr>
<tr><td>Soluble</td><td>BOOLEAN</td></tr>
<tr><td>Attempt</td><td>INTEGER</td></tr>
<tr><td>Version</td><td>CHAR</td></tr>
<tr><td>ProductID</td><td>STRING</td></tr>
</table>
One mark per data type
</td><td>5</td></tr>
</table>

| Question | Answer | Marks |
|---|---|---|
| 2(a) | One mark for each point:<br><br>• Initialise a count to zero<br>• loop 100 times // loop through all of the array<br>• compare an element with "Empty" **in a loop**<br>• increment the count if equal **in a loop**<br>• Output a message together with the count **not inside a loop** | **5** |
| 2(b) | One mark for each point:<br><br>• The breaking down of an algorithm / task / problem<br>• to a level of (sufficient) detail // into smaller parts / sub-tasks<br>• from which it can be programmed // which are easier to program | **3** |

© UCLES 2019

| Question | Answer | Marks |
|---|---|---|
| 2(c) | **Mode**: READ<br>**Description**: Used to read data from a file // input data from a file to the program // only allows you to read data from the file / can't change the data<br><br>**Mode**: APPEND<br>**Description**: Used to add data // write to the end of the text file // output data from the program to the end of a file (without changing / deleting anything)<br><br>One mark for mode; one mark for corresponding description | **4** |
| 2(d) | **Write:**<br>Use an editor to write the source code / program / high-level language code.<br><br>*Or by example of feature:*<br>An editor provides (features such as) context-sensitive prompts / dynamic syntax checking / PrettyPrint / auto-indentation etc.<br><br>**Translate:**<br>A translator (compiler) will convert the source code / program / high-level language code into object code / machine code / an executable file<br><br>A translator (interpreter) is used to translate the source code / program / high-level language code line by line // Or by example: identify syntax errors<br><br>*Or by example of feature:*<br>A translator will identify errors<br><br>**Test:**<br>A debugger is used to find / (help to) correct errors.<br><br>*Or by example of feature:*<br>e.g. single-step, break-points, watch-window…<br><br>One mark per category (Write, Translate, Test) for each reference to a specific 'feature'. | **3** |

| Question | Answer | Marks |
|---|---|---|
| 3(a) | <br>Mark as follows:<br>• One mark for four boxes connected as shown (search, allocate and enable in the correct order)<br>• One mark for each of:<br>  • Arrows labelled AA, BB **and** CC with correct symbols<br>  • Return parameter from Search with correct symbol<br>  • Return boolean from Allocate with correct symbol<br>  • Double-headed arrow for DD<br>• One mark for repetition arrow (either direction) | **6** |
| 3(b) | • He would use his transferrable skills to understand the new program<br>• He could recognise (or equivalent phrase) basic control structures in the language / by example of a program construct (loops, conditional, declaration...)<br>• He could read the comments / meaningful variable names<br>One mark for each bullet point | **2** |

| Question | Answer | Marks |
|---|---|---|
| 4(a) | `DECLARE Name : ARRAY [1:40] OF STRING`<br>`DECLARE Index : INTEGER`<br><br>`FOR Index ← 1 TO 40`<br>`  OUTPUT "Input the name for student ", Index`<br>`  INPUT Name[Index]`<br>`ENDFOR`<br><br>One mark for each of the following:<br><br>1. Declaration of array **and** index<br>2. Loop for 40 elements<br>3. Prompt (as above, including student number) and input for name **in a loop**<br>4. Assign the name to an array element **in a loop** | **4** |
| 4(b) | • Program code easier to read / modify / debug<br>• Easier to access individual elements of / search for a vaue in the 'data set' // single identifier used | **1** |

| Question | Answer | Marks |
|---|---|---|
| 5(a)(i) | To test <u>every path</u> through the code / algorithm<br><br>Accept phrase with equivalent meaning | **1** |
| 5(a)(ii) | A trace table | **1** |
| 5(b)(i) | **String:**<br>• three possible formats for string containg two words:<br>`"Cat∇∇Dog" // "Cat∇Dog∇" // "∇Cat∇Dog"`<br><br>**Explanation:**<br>• When a space character is encountered<br>• `NumWords` is incremented by 1<br><br>**OR:**<br>• The algorithm counts the spaces<br>• and not the words<br><br>1 mark for a string that would give the correct result<br>2 marks for explanation | **3** |

| Question | Answer | Marks |
|---|---|---|
| 5(b)(ii) | **String 1:**<br>• Output: `Number of words : 2`<br><br>**Description 1:**<br>• Check character at end of string<br>• If not a character, increment variable `NumTotal`<br><br>**OR:**<br>• Add a space at the beginning / end of string<br>• At the start of the algorithm<br><br>**OR:**<br>• Change Initialisation of `NumWords`<br>• to 1<br><br>**OR:**<br>• After the loop<br>• Add 1 to `NumWords`<br><br>**String 2:**<br>• Output: `Number of words : 5`<br><br>**Description 2:**<br>• Detect a space followed by a space<br>• Count as a single space / only increment variable `NumTotal` once<br><br>**OR:**<br>• Replace all double spaces with a single space<br>• Before the loop<br><br>**OR:**<br>• After the loop<br>• Subtract 2 from `NumWords`<br><br>Many possible solutions.<br><br>One mark for each correct output<br>One marks for each description bullet point | **6** |

| Question | Answer | Marks |
|---|---|---|
| 6(a) | 'Pseudocode' solution included here for development and clarification of mark scheme.<br><br>Programming language example solutions appear in the Appendix.<br><br>`FUNCTION SearchFile (SearchString : STRING) RETURNS STRING`<br><br>`  DECLARE FileData : STRING`<br>`  DECLARE Found : BOOLEAN`<br>`  DECLARE SearchLength : INTEGER`<br><br>`  Found ← FALSE`<br>`  SearchLength ← LENGTH(SearchString)`<br><br>`  OPENFILE "StudentContact.txt" FOR READ`<br><br>`  WHILE NOT EOF("StudentContact.txt") AND NOT Found`<br>`  READFILE "StudentContact.txt", FileData`<br>`  IF SearchString = LEFT(FileData, SearchLength)`<br>`  THEN`<br>`  Found ← TRUE`<br>`  ENDIF`<br>`  ENDWHILE`<br><br>`  CLOSEFILE "StudentContact.txt"`<br><br>`  IF NOT FOUND`<br>`  THEN`<br>`  RETURN ""`<br>`  ELSE`<br>`  RETURN FileData`<br>`  ENDIF`<br><br>`ENDFUNCTION`<br><br>One mark for each of the following:<br><br>1. Function header and end (where appropriate). Parameter optional but if present must be of type `STRING`<br>2. Calculate length of string from parameter // extract substring from file line<br>3. File `OPEN()` in `READ` mode **and** subsequent `CLOSE()`<br>4. `WHILE` loop repeating until `EOF()`<br>5. read a line from the file **in a loop**<br>6. compare name from file with `SearchString` **in a loop**<br>7. exit loop if `SearchString` found<br>8. Return the line from the file if `SearchString` found **or** an empty string if not found | **8** |

| Question | Answer | Marks |
|---|---|---|
| 6(b) | `FUNCTION ProcessArray() RETURNS INTEGER`<br><br>`DECLARE NoTelNumber : INTEGER`<br>`DECLARE Index : INTEGER`<br>`DECLARE ThisName : STRING`<br>`DECLARE StudentData : STRING`<br><br>`NoTelNumber ← 0`<br><br>`FOR Index ← 1 to 40`<br>`ThisName ← ClassList[Index]`<br>`IF ThisName <> "" //Skip blanks`<br>`THEN`<br>`StudentData ← SearchFile(ThisName)`<br>`IF StudentData = "" //Student not found`<br>`THEN`<br>`StudentData ← ThisName & "*No number"`<br>`NoTelNumber ← NoTelNumber + 1`<br>`ENDIF`<br>`CALL AddToFile(StudentData, "ClassContact.txt")`<br>`ENDIF`<br><br>`ENDFOR`<br><br>`RETURN NoTelNumber`<br><br>`ENDFUNCTION`<br><br>One mark for each of the following:<br><br>1. Function header and end, including return parameter<br>2. Declaration and initialisation of local count variable (`NoTelNumber`)<br>3. `FOR` loop for 40 array elements …<br>4. skip empty elements **in a loop**<br>5. use `SearchFile(ThisName)` and save return value **in a loop**<br>6. if `Searchfile()` returns an empty string, add "*No number" to `SearchString` …<br>7. ... and increment count<br>8. call `AddToFile` with both parameters as above **in a loop**<br>9. Return count **outside the loop** | **9** |
| 6(c) | 'Pseudocode' solution included here for development and clarification of mark scheme.<br><br>Programming language example solutions appear in the Appendix.<br><br>`FUNCTION ProcessArray (ClassList : ARRAY, ClassContact : STRING)`<br>`RETURNS : INTEGER`<br><br>One mark per underlined section. | **3** |

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

**Program Code Example Solutions**

**Q6 (a): Visual Basic**

```
Function SearchFile(SearchString As String) As String
   Dim FileData As String
   Dim Found As Boolean
   Dim SearchLength As Integer
   Dim FileName As String

   Found = False
   SearchLength = Len(SearchString)

   FileName = "StudentContact.txt"
   FileOpen(1, FileName, OpenMode.Input)
   While Not EOF(1) And NOT Found
      FileData = LineInput(1)
      If SearchString = Left(FileData, SearchLength) Then
         Found = True
      End If
   End While

   FileClose(1)

   If Not Found Then
      Return ""
   Else
      Return FileData
   End If

End Function
```

**Alternative:**

```
Function SearchFile (SearchString As String) As String
   Dim FileData As String
   Dim Found As Boolean
   Dim SearchLength As Integer
   Dim MyFile As System.IO.StreamReader

   MyFile = My.Computer.FileSystem.OpenTextFileReader("StudentContact.txt")
   Found = False
   SearchLength = Len(SearchString)

   Do While MyFile.Peek <> -1
      FileData = MyFile.Readline()
      If SearchString = LEFT(FileData, SearchLength) Then
         Found = True
         return(FileData)
      End If
   Loop
   MyFile.Close
   If NOT Found then
      return ("")
   End If
End Function
```

© UCLES 2019

**Q6 (a): Pascal**

```
function SearchFile(var SearchString: string):string;
   var Found : Boolean;
      SearchLength : integer;
      FileData : string;
      MyFile : text;

begin
   Found := False;
   SearchLength := Length(SearchString);

   Assign(MyFile, 'StudentContact.txt');
   Reset(MyFile);

   While NOT EOF(MyFile) AND Found = False do
      Begin
         Readln(MyFile, FileData);
         If SearchString = LeftStr(FileData,SearchLength) then
                  Found := True;
      End;
      Close(MyFile);

   If NOT Found then
      SearchFile := ''
   else
      SearchFile := FileData;

   End;
```

**Q6 (a): Python**

```
def searchFile(searchString):
   ##Declare filedata : string, found : boolean, searchLength : integer
   ##returns a string value

   found = False
   searchLength = len(searchString)
   myFile = open("StudentContact.txt", 'r')
   fileData = myFile.readline()
   while found == False:
      fileData = myFile.readline()
      if not fileData.strip(): #check if no data/end of file
         break
      else:
         if searchString == fileData[0:searchLength]:
            found = True
            print(searchString)

   myFile.close
   if found == False:
      return("")
   else:
      return(fileData)
```

**Q6 (c): Visual Basic**

```
Function ProcessArray (ClassList() As String, ClassContact As String) As Integer
```

**OR**

```
Function ProcessArray (ClassList As String(), ClassContact As String) As Integer
```

**Q6 (c): Pascal**

```
function ProcessArray (var ClassList:CList; ClassContact:string) :integer;
```

CList is user-defined type – could be any name that's not a keyword

**Q6 (c): Python**

```
def ProcessArray (ClassList, ClassContact) :
```

© UCLES 2019