Cambridge International AS & A Level

Cambridge Assessment International Education
Cambridge International Advanced Subsidiary and Advanced Level

COMPUTER SCIENCE **9608/43**

Paper 4 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:

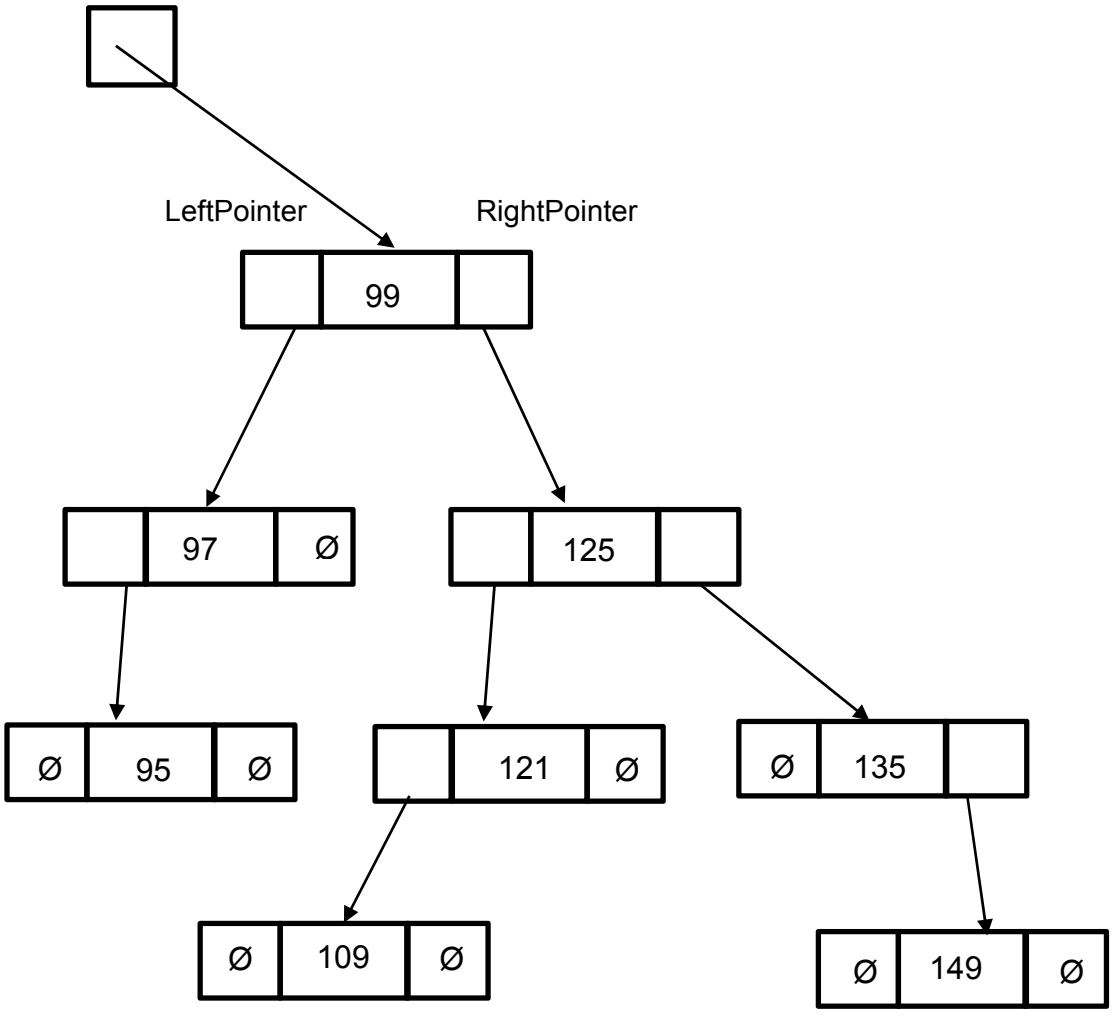
Marks should be awarded using the full range of marks defined in the mark scheme for the question (however; the use of the full mark range may be limited according to the quality of the candidate responses seen).

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)(i)	1 mark for correct stack 	**1**
1(a)(ii)	1 mark for correct stack 	**1**
1(b)	1 mark per bullet point to max 3 • (Linear) data structure • First in First out // FIFO // An item is added to the end of the queue **and** an item is removed from the front • All items are kept in the order they are entered • It has a head pointer and a tail pointer • Can be static or dynamic • A queue can be circular … • …when the (tail) pointer reaches the last position it returns to the first	**3**

Question	Answer	Marks
2(a)(i)	1 mark per bullet point • 95 to left of 97 • 109 to left of 121 • 135 to right of 125 • 149 to right of 135 • Null points in all places and no inappropriate pointers RootPointer  	**5**

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>2(a)(ii)</td><td>1 mark per bullet point

• FreePointer as 8
• 99
• 125
• 121 and 97
• 109 and 95
• 135 and 149

<table>
<tr><th>RootPointer</th></tr>
<tr><td>0</td></tr>
</table>
<table>
<tr><th>FreePointer</th></tr>
<tr><td>8</td></tr>
</table>
<table>
<tr><th>Index</th><th>LeftPointer</th><th>Data</th><th>RightPointer</th></tr>
<tr><td>[0]</td><td>3</td><td>99</td><td>1</td></tr>
<tr><td>[1]</td><td>2</td><td>125</td><td>6</td></tr>
<tr><td>[2]</td><td>4</td><td>121</td><td>null</td></tr>
<tr><td>[3]</td><td>5</td><td>97</td><td>null</td></tr>
<tr><td>[4]</td><td>null</td><td>109</td><td>null</td></tr>
<tr><td>[5]</td><td>null</td><td>95</td><td>null</td></tr>
<tr><td>[6]</td><td>null</td><td>135</td><td>7</td></tr>
<tr><td>[7]</td><td>null</td><td>149</td><td>null</td></tr>
<tr><td>[8]</td><td></td><td></td><td></td></tr>
</table>
</td><td>6</td></tr>
<tr><td>2(b)</td><td>1 mark for each completed section


```
FUNCTION FindElement(Item : INTEGER) RETURNS INTEGER
    CurrentPointer ← RootPointer
    WHILE CurrentPointer <> NullPointer
        IF List[CurrentPointer].Data <> Item
            THEN
                CurrentPointer ← List[CurrentPointer].Pointer
            ELSE
                RETURN CurrentPointer
        ENDIF
    ENDWHILE
    CurrentPointer ← NullPointer
    RETURN CurrentPointer
ENDFUNCTION
```

</td><td>6</td></tr>
<tr><td>2(c)(i)</td><td>1 mark per bullet point to max 3
e.g.
• A sequence of steps that change the state of the program
• The steps are in the order they should be carried out
• e.g. procedural programming/language
• Groups code into self-contained blocks // split the program into modules
• … which are subroutines // by example</td><td>3</td></tr>
</table>

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>2(c)(ii)</td><td>1 mark per bullet point to max 3
e.g.
• Creates classes
• …as a blueprint for an object // objects are instances of classes
• …that have properties/attributes and methods
• … that can be private to the class // properties can only be accessed by the class's methods // encapsulation
• Subclasses can inherit from superclasses (child and parent)
• A subclass can inherit the methods and properties from the superclass
• A subclass can change the methods from the superclass // subclass can use polymorphism
• Objects can interact with each other</td><td>3</td></tr>
<tr><td>2(d)(i)</td><td>1 mark per bullet point

• Method header and close (where appropriate)
• …with <code>InputPlayerID</code> parameter
• Initialise <code>Score</code> to 0
• Initialise Category to <code>"Not Qualified"</code>
• Initialise <code>PlayerID</code> to parameter

PYTHON
<pre>def__init__(self, InputPlayerID):
    self.__Score = 0
    self.__Category = "Not Qualified"
    self.__PlayerID = InputPlayerID</pre>PASCAL
<pre>Constructor Player.Create(InputPlayerID);
begin
    Score := 0 ;
    Category := 'Not Qualified' ;
    PlayerID := InputPlayerID;
end;</pre>VB
<pre>Public Sub New (InputPlayerID)
    Score = 0
    Category = "Not Qualified"
    PlayerID = InputPlayerID
End Sub</pre></td><td>5</td></tr>
</table>

Question	Answer	Marks

2(d)(ii)	1 mark per bullet point
	
	• 1 get Method header without parameter (returning correct data type if given)
	• …returning the property
	• A second working Get
	• A third working Get
	
	PYTHON
	`def GetScore():`
	`  return (Score)`
	`def GetCategory():`
	`  return (Category)`
	`def GetPlayerID():`
	`  return (PlayerID)`
	
	PASCAL
	`function GetScore():Integer;`
	`begin`
	`  GetScore:= Score;`
	`end;`
	`function GetCategory():String;`
	`begin`
	`  GetCategory:= Category;`
	`end;`
	`function GetPlayerID():String;`
	`begin`
	`  GetPlayerID:= PlayerID;`
	`end;`
	
	VB
	`Public Function GetScore() As Integer`
	`  Return Score`
	`End Function`
	`Public Function GetCategory() As String`
	`  Return Category`
	`End Function`
	`Public Function GetPlayerID() As String`
	`  Return PlayerID`
	`End Function`
		4

Question	Answer	Marks

2(d)(iii)	1 mark per bullet point
	
	• Set method header and close (where appropriate)
	• Input value
	• Looping until input value is correct length …
	• … storing **valid** input value in `PlayerID`
	
	PYTHON
	`def SetPlayerID(self)`
	`  PlayerID = input("Enter your player ID")`
	`  while len(PlayerID) > 15 and len(PlayerId) < 4`
	`  PlayerID = input("Must be <=15 AND >=4 characters long. Enter your player ID")`
	
	PASCAL
	`Procedure SetPlayerID ()`
	`  WriteLn ('Enter your player ID');`
	`  ReadLn(PlayerID);`
	`  while length(PlayerID) > 15 and length(PlayerID) < 4 do`
	`  begin`
	`  WriteLn('Must be <=15 AND >=4 characters long. Enter your player ID');`
	`  ReadLn(PlayerID);`
	`  end;`
	
	VB
	`Public Sub SetPlayerID()`
	`  Console.WriteLine ("Enter your player ID")`
	`  PlayerID = Console.ReadLine()`
	`  While Len(PlayerID) > 15 and Len(PlayerID) < 4`
	`  Console.WriteLine ("Must be <=15 AND >=4 characters long. Enter your player ID")`
	`  PlayerID = Console.ReadLine()`
	`  End While`
	`End Sub`
		4

Question	Answer	Marks

2(d)(iv)	1 mark per bullet point
	
	• Function header and close (where appropriate) **and** takes `ScoreInput` as parameter
	• Check if 0 **<=** `ScoreInput` **<=** 150
	• …if valid, set `Score` to parameter
	• …if not valid, output error
	• Returns TRUE if valid **and** returns FALSE if not valid
	
	PYTHON
	`def __SetScore(ScoreInput):`
	`  if ScoreInput >=0 and ScoreInput <=150:`
	`  IsValid = True`
	`  self__Score = ScoreInput`
	`  else:`
	`  print("Error")`
	`  IsValid = False`
	`Return(IsValid)`
	
	PASCAL
	`function Player.SetScore(ScoreInput: Integer) : Boolean;`
	`begin`
	`  If (ScoreInput >=0) AND (ScoreInput <=150) Then`
	`  IsValid := True;`
	`  result := ScoreInput;`
	`  Else`
	`  WriteLn('Error')`
	`  result := False;`
	`end;`
	
	VB
	`Public Function SetScore(ByVal ScoreInput As Integer) As Boolean`
	`  If (ScoreInput >=0) And (ScoreInput <=150) Then`
	`  Return True`
	`  Score = ScoreInput`
	`  Else`
	`  Console.Writeline("Error")`
	`  Return False`
	`  End If`
	`End Function`
		5

Question	Answer	Marks

2(d)(v)	1 mark per bullet point
	
	• Procedure header and close (where appropriate)
	• Accessing Score attribute
	• Correct selection to assign each category
	• … storing in Category attribute
	
	PYTHON
	`def SetCategory()`
	`  if self.__Score >120:`
	`  self.__Category = "Advanced"`
	`  elif self.__Score >80:`
	`  self.__Category = "Intermediate"`
	`  elif self.__Score>=50:`
	`  self.__Category = "Beginner"`
	`  else:`
	`  self.__Category = "Not Qualified"`
	
	PASCAL
	`procedure player.SetCategory()`
	`begin`
	`  If Score >120 Then`
	`  Category := "Advanced";`
	`  Else If Score >80 Then`
	`  Category := "Intermediate";`
	`  Else If Score >= 50 Then`
	`  Category := "Beginner";`
	`  Else`
	`  Category := "Not Qualified";`
	`end;`
	
	VB
	`Public Sub SetCategory()`
	`  If Score >120 Then`
	`  Category = "Advanced"`
	`  ElseIf Score >80 Then`
	`  Category = "Intermediate"`
	`  ElseIf Score >=50 Then`
	`  Category = "Beginner"`
	`  Else`
	`  Category = "Not Qualified"`
	`  End If`
	`End Sub`
		4

Question	Answer	Marks

2(d)(vi)	1 mark per bullet point
	
	• `CreatePlayer()` header and close (where appropriate)
	• Input of `score` and `PlayerID` with suitable prompts
	• Create instance of `Player` named `JoannePlayer` …
	• …with `PlayerID` as parameter
	• Call method `SetScore` for `JoannePlayer` with parameter `Score`
	• …storing return value
	• …outputting appropriate message for not valid
	• Call `SetCategory` for `JoannePlayer`
	• Output Category for `JoannePlayer` …
	• … using `GetCategory` for object Joanne
	
	PYTHON
	`def CreatePlayer():`
	`  InputPlayerID = input("Enter your chosen ID")`
	`  Score = int(input("Please enter the score"))`
	`  JoannePlayer = Player(InputPlayerID)`
	`  if JoannePlayer.SetScore(Score) == false:`
	`  print("Invalid score")`
	`  else:`
	`  JoannePlayer.SetCategory()`
	`  print(JoannePlayer.GetCategory)`
	
	PASCAL
	`procedure CreatePlayer();`
	`  var`
	`  playerID : String;`
	`  isValid : boolean;`
	`  JoannePlayer : Player;`
	`  score : integer;`
	`  begin`
	`  Writeln(Enter Player ID: ');`
	`  Readln(playerID);`
	`  Writeln('Enter score: ');`
	`  Readln(score);`
	`  JoannePlayer := Player.Create(PlayerID);`
	`  isValid := JoannePlayer.SetScore(Score);`
	`  if isValid = true:`
	`  JoannePlayer.SetCategory();`
	`  Writeln(JoannePlayer.GetCategory());`
	`  else:`
	`  Writeln("Invalid score")`
	`end;`
		8

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>2(d)(vi)</td><td>VB
<pre>
Sub CreatePlayer()
   Dim Score As Integer, InputPlayerID As String
   Console.WriteLine("Please enter your chosen ID")
   InputPlayerID = Console.ReadLine()
   Console.WriteLine("Please enter the score")
   Score = Console.ReadLine()

   Dim JoannePlayer As New Player(InputPlayerID)
   if JoannePlayer.SetScore(Score) = True then
      JoannePlayer.SetCategory()
      Console.WriteLine(JoannePlayer.GetCategory())
   else
      Console.Writeline("Invalid score")
   endif
End Sub
</pre></td><td></td></tr>
<tr><td>2(e)</td><td>1 mark per bullet point

• 3 correct Normal test data
• 3 correct Abnormal test data
• 3 correct Boundary test data

<table>
<tr><th>Category</th><th>Type of test data</th><th>Example test data</th></tr>
<tr><td rowspan="3">Beginner</td><td>Normal</td><td>e.g. 75</td></tr>
<tr><td>Abnormal</td><td>e.g. 85 / bob</td></tr>
<tr><td>Boundary</td><td>80, 50</td></tr>
<tr><td rowspan="3">Intermediate</td><td>Normal</td><td>e.g. 95</td></tr>
<tr><td>Abnormal</td><td>e.g. 70 / bob</td></tr>
<tr><td>Boundary</td><td>81, 120</td></tr>
<tr><td rowspan="3">Advanced</td><td>Normal</td><td>e.g. 125</td></tr>
<tr><td>Abnormal</td><td>e.g. 115 / bob</td></tr>
<tr><td>Boundary</td><td>121, 150</td></tr>
</table></td><td>3</td></tr>
<tr><td>2(f)(i)</td><td>Insertion sort</td><td>1</td></tr>
<tr><td>2(f)(ii)</td><td>One from:
• Bubble sort
• Merge sort</td><td>1</td></tr>
</table>

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>2(f)(iii)</td><td>1 mark per shaded section

<table>
<tr><th rowspan="2">Item</th><th rowspan="2">NumberOfScores</th><th rowspan="2">InsertScore</th><th rowspan="2">Index</th><th colspan="5">ArrayData</th></tr>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th></tr>
<tr><td></td><td></td><td></td><td></td><td>99</td><td>125</td><td>121</td><td>109</td><td>115</td></tr>
<tr><td>1</td><td>5</td><td>125</td><td>0</td><td></td><td>(125)</td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td>121</td><td>1</td><td></td><td></td><td>125</td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td>0</td><td></td><td>121</td><td></td><td></td><td></td></tr>
<tr><td>3</td><td></td><td>109</td><td>2</td><td></td><td></td><td></td><td>125</td><td></td></tr>
<tr><td></td><td></td><td></td><td>1</td><td></td><td></td><td>121</td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td>0</td><td></td><td>109</td><td></td><td></td><td></td></tr>
<tr><td>4</td><td></td><td>115</td><td>3</td><td></td><td></td><td></td><td></td><td>125</td></tr>
<tr><td></td><td></td><td></td><td>2</td><td></td><td></td><td></td><td>121</td><td></td></tr>
<tr><td></td><td></td><td></td><td>1</td><td></td><td></td><td>115</td><td></td><td></td></tr>
</table>
</td><td>7</td></tr>
</table>

Question	Answer	Marks
3(a)	1 mark per bullet point to max 2 • It is defined in terms of itself // it calls itself • It has a stopping condition // base case • It is a self-contained subroutine • It can return data to its previous call	2
3(b)	1 mark per bullet point to max 3 • (When the recursive call is made) all values/data are put on … • … the stack • When the stopping condition/base case is met • …the algorithm unwinds • …the last set of values are taken off the stack (in reverse order)	3