Cambridge International AS & A Level

Cambridge Assessment International Education
Cambridge International Advanced Subsidiary and Advanced Level

COMPUTER SCIENCE **9608/42**

Paper 4 Written Paper **October/November 2018**

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 2018 series for most Cambridge IGCSE™, Cambridge International A and AS Level components and some Cambridge O Level components.

This document consists of **18** printed pages.

© UCLES 2018

Cambridge Assessment International Education

[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.

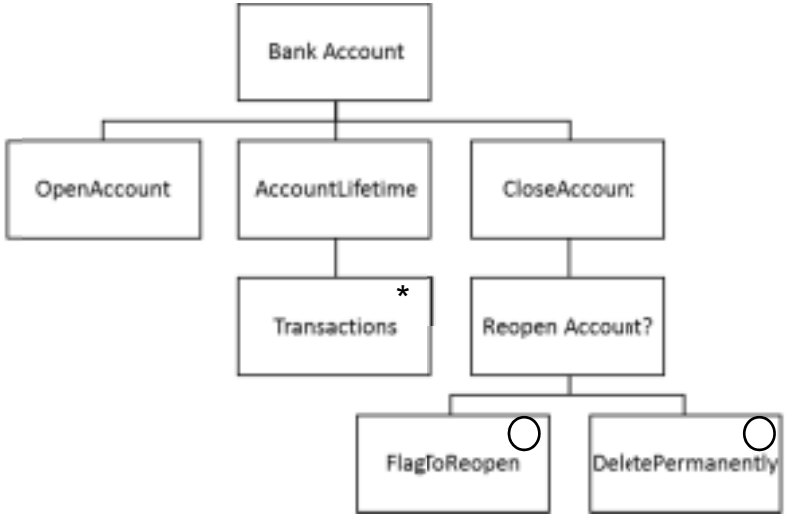
GENERIC MARKING PRINCIPLE 4:

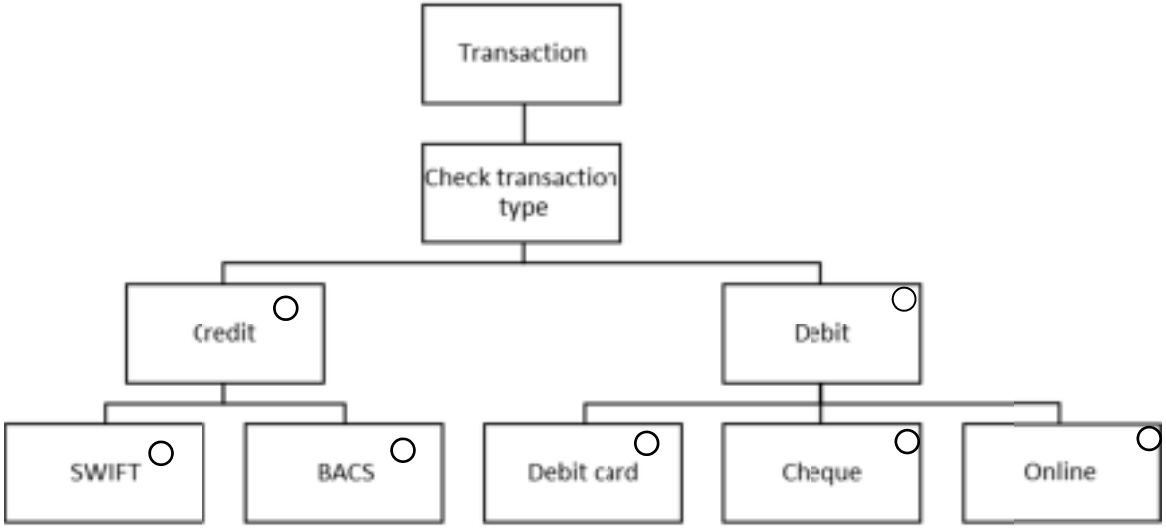
Rules must be applied consistently e.g. in situations where candidates have not followed instructions or in the application of generic level descriptors.

© UCLES 2018

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 2018

Question	Answer	Marks
1(a)	1 mark per bullet point: • `OpenAccount`, `AccountLifetime` and `CloseAccount` below `Bank Account Transactions` below `AccountLifetime` • …with iteration • `Reopen Account?` below `Close Account` • `FlagToReopen` and `DeletePermanently` below `ReopenAccount?` • …with selection on both Example:  	6

Question	Answer	Marks
1(b)	1 mark per bullet point: • `Credit` and `debit` below `Transaction` • `Swift` and `BACS` below `Credit` • `Debit`, `Cheque` and `Online` below `Debit` • Correct selections where needed and no additional selection/iteration  	**4**

Question	Answer	Marks

2(a)	1 mark for each fact:
	
	`type(waterdog, gundog).`
	`is_a(standardpoodle, waterdog).`
		2

2(b)	1 mark for each result: `H = english_setter, irish_setter`	**2**

Question	Answer	Marks
2(c)	1 mark per bullet point to max 2: • `is_a` • `(irish_setter, W)` `is_a(irish_setter, W)`	**2**
2(d)	1 mark per bullet point to max 3: • `is_a(X,Z)` • `AND` • `fav_bird(Z, Y).` `fav_bird(X, Y) IF is_a(X, Z) AND fav_bird(Z, Y).`	**3**
2(e)	NO	**1**

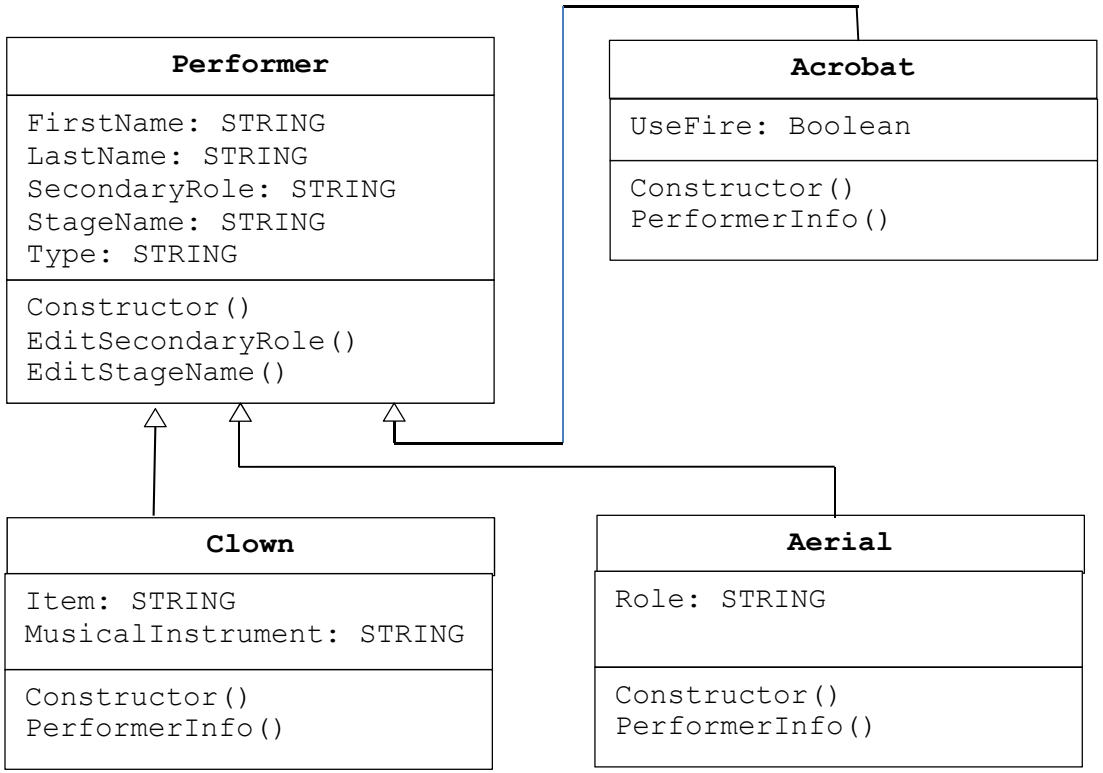
Question	Answer	Marks

3(a)	1 mark for each completed statement:
	
	`FOR Outer ← LENGTH(List)-1 TO 0 STEP -1`
	`  FOR Inner ← 0 TO (Outer - 1)`
	`  IF List[Inner] > List[Inner + 1]`
	`  THEN`
	`  Temp ← List[Inner]`
	`  List[Inner] ← List[Inner + 1]`
	`  List[Inner + 1] ← Temp`
	`  ENDIF`
	`  ENDFOR`
	`ENDFOR`
		7

3(b)(i)	Ascending (must match answer to 3(a)	**1**

Question	Answer	Marks
3(b)(ii)	Line 03 Change the operator in the IF statement to < or <= rather than >	**1**

3(c)	1 mark per bullet
	
	• Use of a (Boolean) flag…
	• …Remainder of bubble correct
	• Set flag when a swap has been made…
	• …Loop until a swap has not been made **and** then exit all loops
	
	`Outer ← LENGTH(List)-1`
	`REPEAT`
	`  Inner ← 0`
	`  Swap ← FALSE`
	`  REPEAT`
	`  IF List[Inner] > List[Inner + 1]`
	`  THEN`
	`  Temp ← List[Inner]`
	`  List[Inner] ← List[Inner + 1]`
	`  List[Inner + 1] ← Temp`
	`  Swap = TRUE`
	`  ENDIF`
	`  Inner ← Inner + 1`
	`  UNTIL Inner = Outer - 1`
	`  Outer ← Outer - 1`
	`UNTIL Swap = FALSE OR Outer = 0`
		4

Question	Answer	Marks
4(a)	1 mark per bullet point: • `Clown` has attributes `Item` and `MusicalInstrument` with appropriate data types • `Aerial` has attribute `Role/Type` with appropriate data type. • `Clown` **and** `Aerial` have method `PerformerInfo()/DisplayInfo()` or equivalent. • Acrobat, Clown and Aerial inheriting from Performer.  	**4**

Question	Answer	Marks

4(b)	1 mark per bullet point to max 5:
	
	1 mark per bullet to max 4:
	• class declaration (and end where applicable)
	• declaring **five** attributes as private (with string data types where applicable)
	• (language specific) constructor method (and end where applicable)
	• … with **five** parameters
	• initialising **five** attributes … … using parameters
	
	1 mark per bullet to max 3:
	• procedure header (and close) for `EditSecondaryRole` // procedure header (and close) for `EditStageName` …
	• …takes parameter
	• …`EditSecondaryRole` replaces `SecondaryRole` with parameter
	• …`EditStageName` replaces `StageName` with parameter
	
	Example **Python**:
	
	`class Performer(object):`
	
	`def __init__(self, Firstname, Lastname, Stagename, SecondaryRole, Type):`
	`self.__FirstName = Firstname`
	`self.__LastName = Lastname`
	`self.__StageName = Stagename`
	`self.__SecondaryRole = SecondaryRole`
	`self.__PerfType = Type`
	
	`def EditSecondaryRole(self, NewRole):`
	`self.SecondaryRole = NewRole`
	
	`def EditStageName(self, NewStageName):`
	`self.StageName = NewStageName`
		5

Question	Answer	Marks

4(b)	Example **Visual Basic**:
	
	`Class Performer`
	`  Private  FirstName As String`
	`  Private  LastName As String`
	`  Private  StageName As String`
	`  Private  SecondaryRole As String`
	`  Private  PerfType As String`
	
	`  Public Sub New(ByVal Fname As String,ByVal Lname As String,`
	`  ByVal Sname As String, ByVal SecRole As String, ByVal Type As`
	`  String)`
	`  FirstName = Fname`
	`  LastName = Lname`
	`  StageName = Sname`
	`  SecondaryRole = SecRole`
	`  PerfType = Type`
	
	`  End Sub`
	
	`  Public Sub EditSecondaryRole(ByVal Srole As String)`
	`  SecondaryRole = Srole`
	`  End Sub`
	
	`  Public Sub EditStageName(ByVal Sname As String)`
	`  StageName = Sname`
	`  End Sub`
	
	`End Class`	

Question	Answer	Marks

4(b)	Example **Pascal**:
	
	`type Performer = class`
	
	`  private`
	`  FirstName : String;`
	`  LastName : String;`
	`  StageName : String;`
	`  SecondaryRole : String;`
	`  PerfType : String;`
	`  public`
	`  Constructor init(Fname, Lname, Sname, SType, Srole: String);`
	`  Procedure EditSecondaryRole(Srole: String);`
	`  Procedure EditStageName(Sname: String);`
	`  end;`
	
	`Constructor Performer.init(Fname, Lname, Sname, Stype, Srole:String);`
	`begin`
	`  Firstname := Fname;`
	`  Lastname := Lname;`
	`  StageName := Sname;`
	`  SecondaryRole := Srole;`
	`  PerfType := Stype;`
	`end;`
	
	`Procedure Performer.EditSecondaryRole(Srole: String);`
	`begin`
	`  SecondaryRole := Srole;`
	`end;`
	
	`Procedure Performer.EditStageName(Sname: String);`
	`begin`
	`  StageName := Sname;`
	`end;`	

Question	Answer	Marks

4(c)	1 mark per bullet point to max 8:
	
	• class declaration with inheritance from `Performer`
	• constructor taking five or six parameters
	• …call to inherited constructor …
	• …sending either five parameters **or** four with "Acrobat
	• `UseFire` declared as private `Boolean`
	• In constructor, storing value in `UseFire` from parameter
	• `PerformerInfo` header (and end where applicable) without any parameters
	• … outputs / returns
	• … `StageName & " (real name " & FirstName & " " & SecondName & ") is an acrobat"`
	• … `"Fire is part of " & StageName & "'s act."` ONLY printed when Fire is TRUE
	• … `"When not performing, " & StageName & " is a " & SecondaryRole`
	
	Example **Python**:
	
	`class Acrobat(Performer):`
	
	`def __init__(self,Firstname, Lastname, Stagename, SecondaryRole, Fire):`
	`Performer.__init__(self, Firstname, Lastname, Stagename, SecondaryRole, "Acrobat")`
	
	`self.__UseFire = Fire`
	
	`def PerformerInfo(self):`
	`ReturnString = "%s (real name %s %s) is %s. " % (self. Stagename, self.Firstname, self. Lastname, Acrobat.PerfType)`
	`if(self.__UseFire):`
	`ReturnString = ReturnString + "Fire is part of %s's act. " % (self.Stagename)`
	`else:`
	`ReturnString = ReturnString + "Fire is not part of %s's act. " % (self.Stagename)`
	
	`ReturnString = ReturnString + "When not performing, %s is a %s" % (self.Stagename, self.SecondaryRole)`
	`return ReturnString`
		8

Question	Answer	Marks

4(c)	Example **Visual Basic**:
	
	`Class Acrobat : Inherits Performer`
	`  Private UseFire As Boolean`
	
	`  Public Function PerformerInfo() as string`
	`  PerformerInfo = Stagename + "(real name " + FirstName + " " + LastName + ") is "`
	`  + PerfType + "."`
	
	`  IF(UseFire) THEN`
	`  PerformerInfo = PerformerInfo + "Fire is part of " + Stagename + "'s act."`
	`  ELSE`
	`  PerformerInfo = PerformerInfo + "Fire is not part of " + Stagename + "'s act."`
	`  END IF`
	`  PerformerInfo = PerformerInfo + "When not performing, " +  Stagename + " is a`
	`  " + SecondaryRole`
	`  END FUNCTION`
	
	`  Public Sub New(ByVal Fname As String,ByVal Lname As String, ByVal Sname As String,`
	`  ByVal SecRole As String, ByVal fire As String)`
	`  MyBase.New(Fname, Lname, Sname, SecRole, "Acrobat")`
	`  UseFire = fire`
	`  End Sub`
	
	`End Class`	

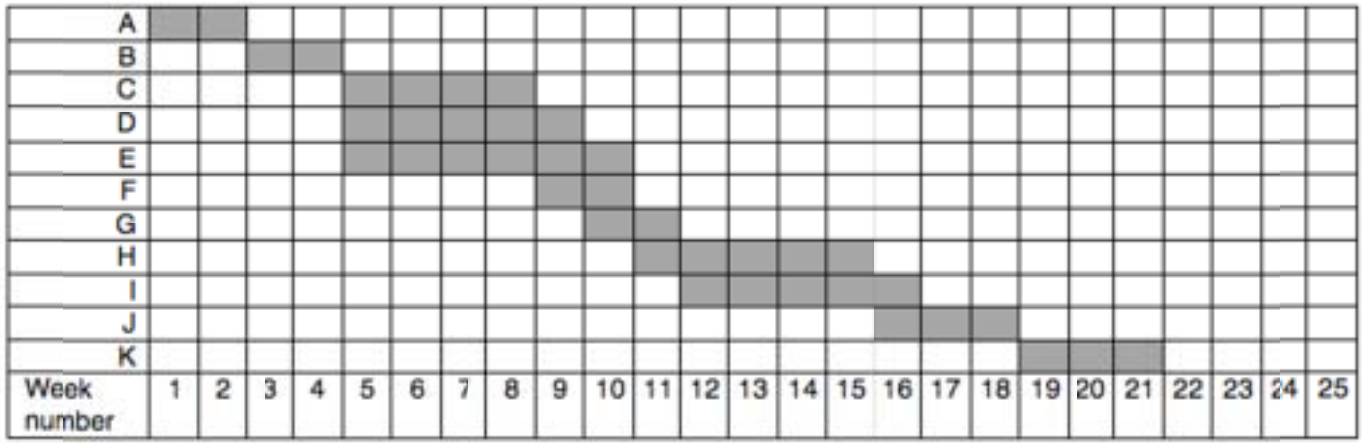
Question	Answer	Marks
4(c)	Example **Pascal**:	

```
type   Acrobat = class(Performer)
    private
       UseFire : Boolean;
    public
     Constructor init(Fname, Lname, Sname, Sfire, Srole: String, "Acrobat"); override;
     Function PerformerInfo() : String
    end;

constructor Acrobat.init(Fname, Lname, Sname, Sfire, Srole:String);
begin
   Firstname := Fname;
   Lastname := Lname;
   StageName := Sname;
   SecondaryRole := Srole;
   PerfType := "Acrobat";
   UseFire := Sfire;
end;

Function Acrobat.PerformerInfo() : String;
  var ReturnString : String;
  begin
    ReturnString := Stagename + "(real name" + FirstName + " " + LastName + ") is " +
                    PerfType;

    IF(UseFire) THEN
      ReturnString := ReturnString + "Fire is part of " + Stagename + "s act."
    ELSE
      ReturnString := ReturnString + "Fire is not part of " + Stagename + "'s act. ";

    ReturnString := ReturnString + "When not performing," +  Stagename + " is a " +
                    SecondaryRole;

    Result = ReturnString;

  end;
```

Question	Answer	Marks

4(d)(i)	1 mark per bullet point:
	
	• Assignment to `Acrobat_1`
	• Creates instance of `Acrobat`
	• **Correct five** parameter values
	
	Example **Python**:
	
	`Acrobat_1 = Acrobat("Alex","Tan","Amazing Alex","Popcorn Seller",`
	`True)`
	
	Example **VB.NET**:
	
	`Acrobat_1 = New Acrobat("Alex","Tan","Amazing Alex","Popcorn Seller",`
	`True)`
	
	Example **Pascal**:
	
	`Acrobat_1 := Acrobat("Alex","Tan","Amazing Alex","Popcorn Seller",`
	`True)`
		3

4(d)(ii)	1 mark per bullet point to max 2: • Clown/Acrobat/Aerial **inherit** from Performer/base class // Clown/Acrobat/Aerial are the child/sub class **and** Performer in the parent/base/super • Clown/Acrobat/Aerial can use the **attributes** from Performer • Clown/Acrobat/Aerial can use the **methods** from Performer • Clown/Acrobat/Aerial can **extend** the **methods** in Performer	**2**

Question	Answer	Marks
5(a)	1 mark per bullet • C, D and E start at same point after B • F follows C • G follows D **and** H follows F • I follows G **and** J follows H • K follows J  	**5**
5(b)	1 mark per bullet point to max 2: Example: • Teams can work on simultaneous/concurrent/parallel tasks • Any example of two teams working simultaneously e.g. From step 3, each team can work on a different task in week 4, 5 and 6 // tasks C, D, E can be split between different teams • Any example of working on same activities //e.g. all teams can work together on 1–2 / A, 2–3/B • Any example of working on dependent activities // e.g. if all teams work on B, then they can split up for CDE	**2**
5(c)(i)	A,B,E,H,J,K	**1**

Question	Answer	Marks
5(c)(ii)	1 mark per bullet point to max 2: Example: • If there is any delay to a task which is part of the critical path • … then the overall project will be delayed • Gives the earliest possible completion time • … this allows you to organise/allocate resources (efficiently) • Frequently recalculate the critical path … • … to see if there are any delays/new critical path has arisen • Can identify where there is slack in activities • … so they can start later without affecting the critical path	**2**

Question	Answer	Marks

6(a)(i)	1 mark per bullet point:
	
	• Declaring a record type // class etc.
	• Declaring Country as a string
	• Declaring Pointer as an integer
	
	Example:
	
	`TYPE ListElement`
	`  DECLARE Country : STRING`
	`  DECLARE Pointer : INTEGER`
	`ENDTYPE`
		3

Question	Answer	Marks
6(a)(ii)	1 mark per bullet point: • Declaring `CountryList` as an array with 15 elements • Of type `ListElement` Example: `DECLARE CountryList : ARRAY[1 : 15] OF ListElement`	**2**

6(b)	1 mark for each completed statement
	`PROCEDURE DeleteNode(NodeValue: STRING, ThisPointer: INTEGER, PreviousPointer: INTEGER)`
	`  IF CountryList[ThisPointer].Value = NodeValue`
	`  THEN`
	`  CountryList[ThisPointer].Value ← ""`
	`  IF ListHead = ThisPointer`
	`  THEN`
	`  ListHead ← CountryList[ThisPointer].Pointer`
	`  ELSE`
	`  CountryList[PreviousPointer].Pointer ← CountryList[ThisPointer].Pointer`
	`  ENDIF`
	`  CountryList[LastNode].Pointer ← ThisPointer`
	`  LastNode ← ThisPointer`
	`  CountryList[ThisPointer].Pointer ← -1`
	`  ELSE`
	`  IF CountryList[ThisPointer].Pointer <> -1`
	`  THEN`
	`  CALL DeleteNode(NodeValue,CountryList[ThisPointer].Pointer, ThisPointer)`
	`  ELSE`
	`  OUTPUT "DOES NOT EXIST"`
	`  ENDIF`
	`  ENDIF`
	`ENDPROCEDURE`
		5