Cambridge International AS & A Level

Cambridge Assessment International Education
Cambridge International Advanced Subsidiary and Advanced Level

COMPUTER SCIENCE **9608/43**

Paper 4 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 **19** printed pages.

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:

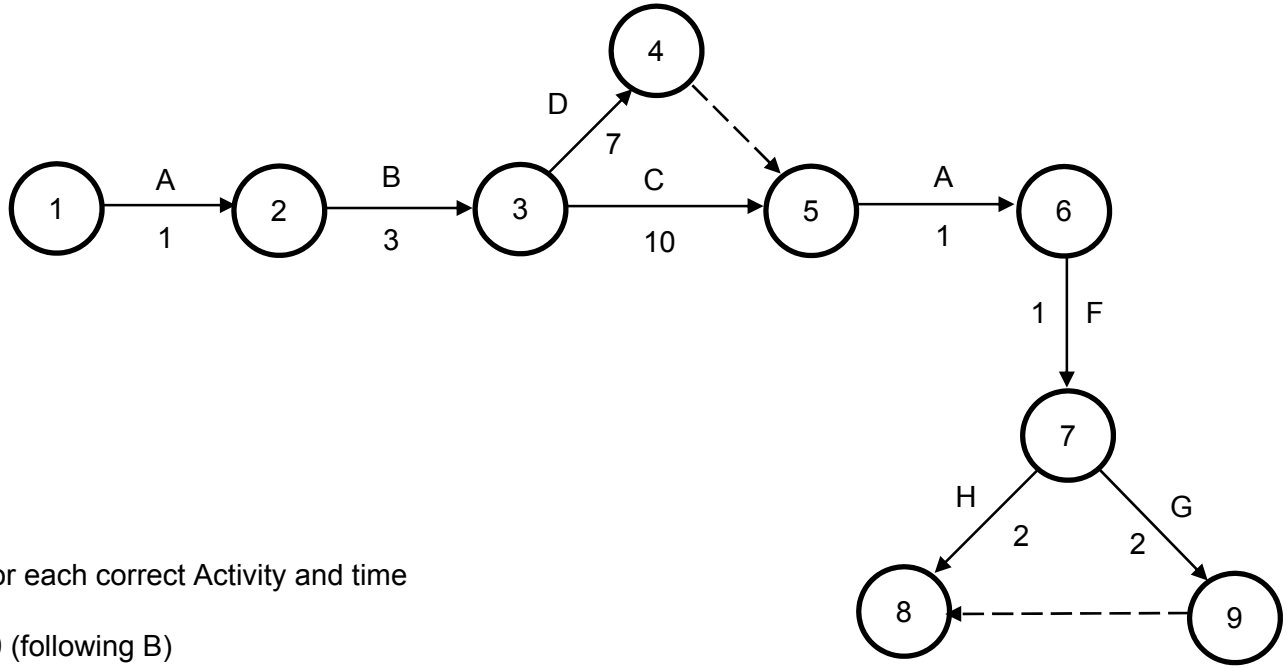
Marks must be awarded **positively**:

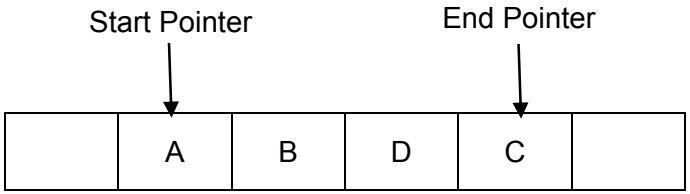
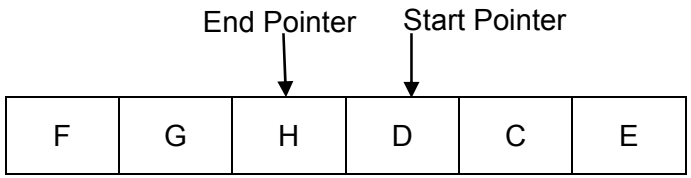
- marks are awarded for correct/valid answers, as defined in the mark scheme. However, credit is given for valid answers which go beyond the scope of the syllabus and mark scheme, referring to your Team Leader as appropriate
- marks are awarded when candidates clearly demonstrate what they know and can 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.

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.

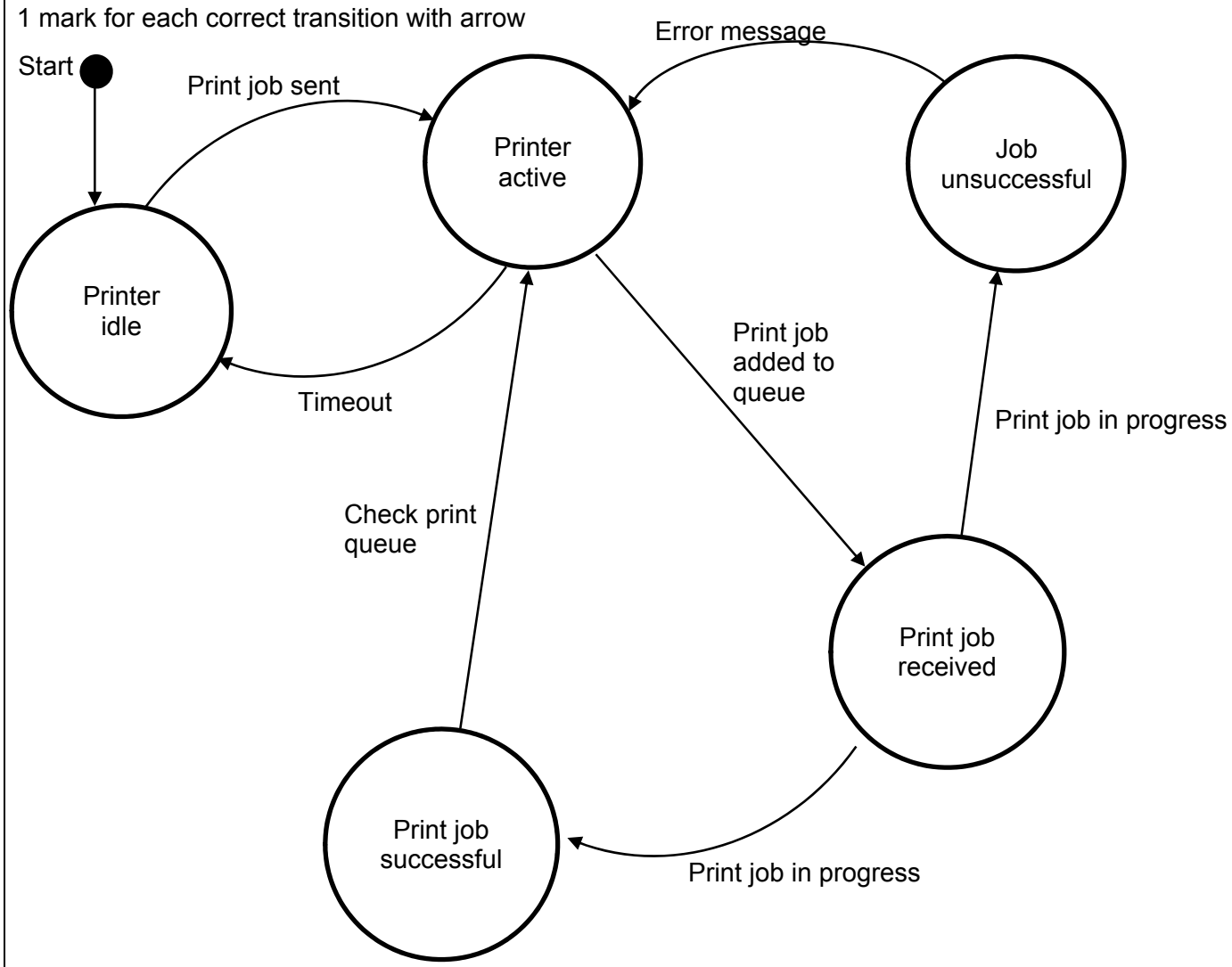
Question	Answer	Marks
1(a)(i)	 1 mark for each correct Activity and time • B 3 • C 10 (following B) • D 7 (following B) • E 3 (following C and D) • G 2 (following F) • H 2 (following F)	**6**
1(a)(ii)	The shortest time to complete the project // the sequence of activities that must be completed to avoid delaying the project	**1**
1(a)(iii)	1 mark for identify, max 1 for description • GANTT • A table that has time across the top and activities on the left, boxes are coloured to show dependencies and find critical path // colour in the boxes to show the length of time for each task	**2**

Question	Answer	Marks
1(b)(i)	A, B, C and D in correct places with no alteration to start and end pointer  	**1**
1(b)(ii)	1 mark per bullet point • correct jobs in correct order… • … correct location of start pointer • … correction location of new end pointer  	**3**
1(b)(iii)	1 mark from: • An error **message** would be generated	**1**

Question	Answer	Marks

1(b)(iv)	1 mark for each correct line
	
	`FUNCTION Remove RETURNS STRING`
	`  DECLARE PrintJob : STRING`
	`  IF StartPointer = EndPointer`
	`  THEN`
	`  RETURN "Empty"`
	`  ELSE`
	`  PrintJob ← Queue[StartPointer]`
	`  IF StartPointer = 5`
	`  THEN`
	`  StartPointer ← 0`
	`  ELSE`
	`  StartPointer ← StartPointer + 1`
	`  ENDIF`
	`  RETURN PrintJob`
	`  ENDIF`
	`ENDFUNCTION`
		4

1(b)(v)	1 mark per bullet point • A stack is Last In First Out (LIFO) while a queue is First In First Out (FIFO) • The queue removes and returns the element at **start pointer** // item is removed from the **start/head //** • A stack would remove and return the element at **end pointer** // item is removed from the **end**	**2**

Question	Answer	Marks
1(c)	1 mark for each correct transition with arrow  	5

Question	Answer	Marks
1(d)(i)	1 mark per bullet • Way of modelling logic • Show **all** possible outputs // shows **every** possible outcome // **all** outcomes … • … based on the inputs • Determine which action to take in specific conditions // how different conditions affect the actions/outcomes	**2**
1(d)(ii)	1 mark for each row Accept Y/X/ticks as long as clear which are Y. Accept N/X/– for empty spaces	**4**

		Rules							
Conditions	Document printed, but quality is poor	Y	Y	Y	Y	N	N	N	N
	Error light is flashing on printer	Y	Y	N	N	Y	Y	N	N
	Document printed, but paper size is incorrect	Y	N	Y	N	Y	N	Y	N
Actions	Check connection from computer to printer						**X**		
	Check ink status	**X**	**X**	**X**	**X**				
	Check if there is a paper jam						**X**		
	Check paper size selected	**X**		**X**		**X**		**X**	

Question	Answer	Marks
1(d)(iii)	1 mark each for each correct column Accept –/X for empty spaces. Accept Y/X/Ticks as long as clear which are used	5

		Rules							
Conditions	Document printed, but quality is poor	**Y**	**Y**	**N**	**N**	**N**			
	Error light is flashing on printer				**Y**	**N**			
	Document printed, but paper size is incorrect	**Y**	**N**	**Y**	**N**	**N**			
Actions	Check connection from computer to printer				**X**				
	Check ink status	**X**	**X**						
	Check if there is a paper jam				**X**				
	Check paper size selected	**X**		**X**					

Question	Answer	Marks

1(e)(i)	1 mark per bullet point to **max 4**
	
	• Method header and close (where necessary) with three parameters
	• Initialised `PrintID`, `FirstName`, `LastName` and `Credits` …
	• … to the parameters
	• Initialised `Credits` to 50
	
	PYTHON
	`def__init__(self, NewFN, NewLN, NewPrintID):`
	`  self.__PrintID = NewPrintID`
	`  self.__FirstName = NewFN`
	`  self.__LastName = NewLN`
	`  self.__Credits = 50`
	
	PASCAL
	`Constructor NewPrintAccount.Create(NewFN, NewLN, NewPrintID);`
	`begin`
	`  PrintID := NewPrintID;`
	`  FirstName = NewFN;`
	`  LastName = NewLN;`
	`  Credits := 50;`
	`end;`
	
	VB
	`Public Sub New(NewFN, NewLN, NewPrintID As String)`
	`  PrintID = NewPrintID`
	`  FirstName = NewFN`
	`  LastName = NewLN`
	`  Credits = 50`
	`End Sub`
		4

Question	Answer	Marks

1(e)(ii)	1 mark per bullet point
	
	• method/procedure header (and close where appropriate) taking a parameter
	• `FirstName` is set to **parameter**
	
	PYTHON
	`def __SetFirstName(self, NewFirstName):`
	`  self.__FirstName = NewFirstName`
	
	PASCAL
	`procedure SetFirstName(newFirstName : String);`
	`begin`
	`  FirstName := newFirstName;`
	`end;`
	
	VB
	`public sub SetFirstName(NewFirstName As String)`
	`  FirstName = NewFirstName`
	`End Sub`
		2

Question	Answer	Marks

1(e)(iii)	1 mark per bullet point
	
	• concatenates `FirstName`, space and `LastName` …
	• … function/method header without parameter **and** returns (generated) value
	
	PYTHON
	`def __GetName(self):`
	`  return(self.__FirstName + " " + self.__LastName)`
	
	PASCAL
	`function GetName();`
	`begin`
	`  result := FirstName + " " + LastName`
	`end;`
	
	VB
	`public function GetName() As String`
	`  return(FirstName & " " & LastName)`
	`End Function`
		2

Question	Answer	Marks

1(e)(iv)	1 mark per each correct bullet point from:
	• **Procedure**/method header and close (where necessary) passing MoneyInput
	• At least **3** constants (e.g. `freecredit10`, `freecredit20`, `twenty`, `10`, `creditperdollar`)
	• If `MoneyInput<10` calculate `MoneyInput * 25`
	• If `MoneyInput >9` and `MoneyInput <20` then calculate `MoneyInput * 25 + 25`
	• If `MoneyInput >19` then calculate `MoneyInput * 25 + 50`
	• All **three** correct calculations add to `Credits`, not overwrite
	• Efficient IF (i.e. elseif)
	
	PYTHON
	`def __AddCredits(self, MoneyInput):`
	`  CreditPerDollar = 25`
	`  FreeCredit10 = 25`
	`  FreeCredit20 = 50`
	`  Twenty = 20`
	`  Ten = 10`
	`  if MoneyInput >= Twenty:`
	`  Credits = Credits + (MoneyInput * CreditPerDollar) + FreeCredit20`
	`  elif MoneyInput >= Ten:`
	`  Credits = Credits + (MoneyInput * CreditPerDollar) + FreeCredit10`
	`  else:`
	`  Credits = Credits + (MoneyInput * CreditPerDollar)`
		6

Question	Answer	Marks

1(e)(iv)	PASCAL
	`procedure AddCredits(MoneyInput : Real);`
	`  const CreditPerDollar = 25`
	`  const FreeCredit10 = 25`
	`  const FreeCredit20 = 50`
	`  const Twenty = 20`
	`  const Ten = 10`
	`begin`
	`  If MoneyInput > = Twenty Then`
	`  Credits := Credits + (MoneyInput * CreditPerDollar) + FreeCredit20;`
	`  Else If MoneyInput > = Ten Then`
	`  Credits := Credits + (MoneyInput * CreditPerDollar) + FreeCredit10;`
	`  Else`
	`  Credits := Credits + (MoneyInput * CreditPerDollar);`
	`end;`
	
	VB.NET
	`Public Sub AddCredits(MoneyInput As Integer)`
	`  Const CreditPerDollar As Integer = 25`
	`  Const FreeCredit10 As Integer = 25`
	`  Const FreeCredit20 AS integer = 50`
	`  Const Twenty As Integer = 20`
	`  Const Ten AS Integer = 10`
	`  If MoneyInput > = Twenty Then`
	`  Credits = Credits + (MoneyInput * CreditPerDollar) + FreeCredit20`
	`  Else If MoneyInput > = Ten Then`
	`  Credits = Credits + (MoneyInput * CreditPerDollar) + FreeCredit10`
	`  Else`
	`  Credits = Credits + (MoneyInput * CreditPerDollar)`
	`  End If`
	`End Sub`	

Question	Answer	Marks
1(e)(v)	1 mark per bullet • Declaring `StudentAccounts` as array of 1000 elements… • …of type `PrintAccount` `DECLARE StudentAccounts ARRAY[0:999] OF PrintAccount`	**2**

Question	Answer	Marks

1(e)(vi)	1 mark per bullet point to **max 8**
	
	• Generating ID with '1' at the end all in **lowercase** from parameters
	
	• Loop through array to last occupied element …
	• … Check if the `PrintID` already exists
	• … using `GetPrintID()`
	• … increment number at end of `PrintID`
	
	• Create a new instance of `PrintAccount` …
	• … sending `FirstName`, `LastName`, `PrintID` as parameters
	• adding new account to `StudentAccounts` at position `NumberStudents`
	• Increment `NumberStudents`
	
	VB.NET
	`Sub CreateId(firstName, lastName)`
	`  Dim count As Integer`
	`  Dim PrintID = Left(firstname, 3).ToLower & Left(lastname, 3).ToLower & "1"`
	`  Dim studentAdd As Integer = 0`
	`  If numberStudents <> 0 Then`
	`    For x = 0 To numberStudents - 1`
	`      If studentAccounts(x).getPrintID() = username Then`
	`        PrintID = PrintID + 1`
	`        username = Left(firstname, 3).ToLower & Left(lastname, 3).ToLower &`
	`PrintID.ToString`
	`       End If`
	`     Next`
	`     studentAdd = numberStudents`
	`   End If`
	`   studentAccounts(studentAdd) = New printAccount(firstname, lastname, username)`
	`   numberStudents = numberStudents + 1`
		8

Question	Answer	Marks

1(e)(vi)	**Python**
	`def CreateID(firstname, lastname):`
	`  count = 0`
	`  PrintID = firstname[0:3].lower() + lastname[-3].lower() + "1"`
	`  StudentAdd = 0`
	`  if numberStudents != 0:`
	`  for x in range(0, numberStudents):`
	`  if studentAccounts[x].getPrintID() == username:`
	`  PrintID = PrintID + 1`
	`  username = firstname[0:3].lower() + lastname[0:3].lower + str(PrintID)`
	`  studentAdd = numberStudents`
	`  studentAccounts[studentAdd] = printAccount(firstname, lastname, username)`
	`  numberStudents = numberStudents + 1`
	
	Pascal
	`procedure CreateID(firstname : String, lastname: String);`
	`  var`
	`  count : Integer;`
	`  studentAdd : Integer;`
	`  PrintID : String;`
	
	`begin`
	`  studentAdd := 0;`
	`  PrintID := LowerCase(substr(firstname,0,3)) + LowerCase(substr(lastname,0,3))+ "1";`
	`  if numberStudents <> 0:`
	`  ror x := 0 To numberStudents - 1;`
	`  if studentAccounts[x].getPrintID() = username:`
	`  PrintID := PrintID + 1;`
	`  username := LowerCase(substr(firstname, 3) +`
	`LowerCase(substr(lastname,0,3))+str(PrintID);`
	
	`  studentAdd := numberStudents;`
	`studentAccounts[studentAdd] := printAccount.Create(firstname, lastname, username);`
	`  numberStudents := numberStudents + 1`	

Question	Answer	Marks
2	1 mark for each highlighted section	9

Label	Op Code	Operand	Comment	
LOOP:	LDD	ANSWER	// Load the value from ANSWER	
	ADD	NUMONE	// Add the value from NUMONE	[1]
	STO	ANSWER		[1]
	LDD	COUNT	// Load the value from COUNT	[1]
	INC	ACC	// Increment the Accumulator	[1]
	STO	COUNT		[1]
	CMP	NUMTWO	// Is NUMTWO = COUNT?	[1]
	JPN	LOOP	// If false, jump to LOOP	[1]
	LDD	ANSWER	// Load the value from ANSWER	[1]
	OUT		// output ANSWER to the screen	[1]
	END		// End of program	
NUMONE:	2			
NUMTWO:	4			
COUNT:	0			
ANSWER:	0			

Question	Answer	Marks
3	1 mark per bullet point to **max 3** • Logic error // it is programmed incorrectly • There was an error in the design // the correct requirements were not stated • Run-time error // division by 0 // stack overflow // end of file reached // library not available // linking/loading error • Not adequately/correctly tested	**3**

Question	Answer	Marks
4	1 mark for each term.	**3**

Definition	Term
Software is tested by an in-house team of dedicated testers.	**Alpha testing/black-box/white-box**
Software is tested by the customer before it is signed off.	**Acceptance testing**
Software is tested by a small selection of users before general release	**Beta testing**