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 2017**

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.

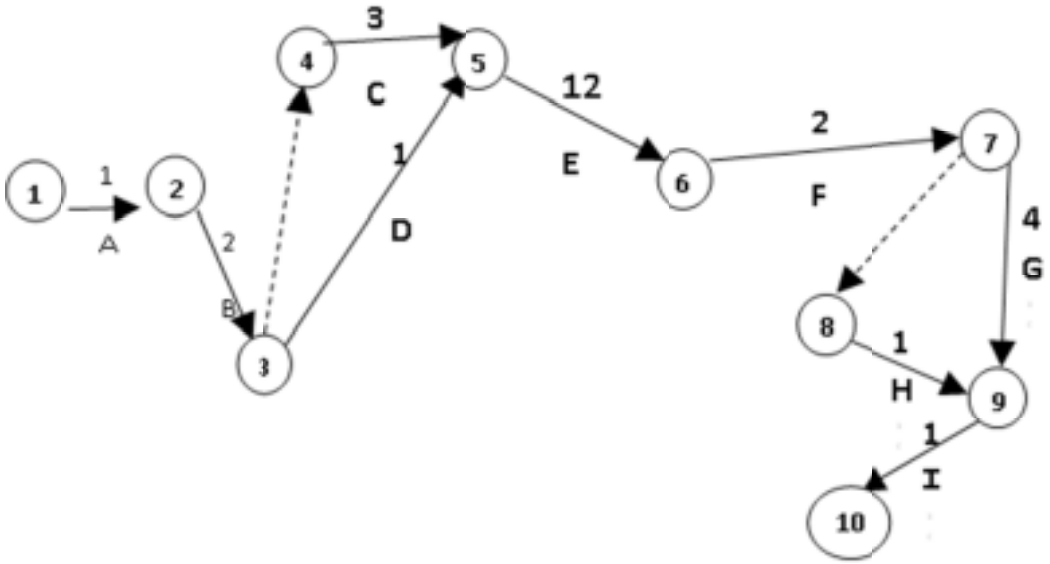
Cambridge International will not enter into discussions about these mark schemes.

Cambridge International is publishing the mark schemes for the October/November 2017 series for most Cambridge IGCSE®, Cambridge International A and AS Level components and some Cambridge O Level components.

Question	Answer	Marks
1(a)	1 mark per shaded group	**4**

		Column							
		1	2	3	4	5	6	7	8
Conditions	Grade C in Computer Science	Y	Y	Y	Y	N	N	N	N
	Grade C in Maths	Y	Y	N	N	Y	Y	N	N
	Grade C in Science	Y	N	Y	N	Y	N	Y	N
Actions	Take Computer Science	Y	Y	Y	Y	Y	Y		
	Take Maths	Y	Y			Y	Y		
	Take Physics	Y				Y			

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>1(b)</td><td>1 mark per column
<table>
<tr><td colspan="2"></td><td colspan="8">Column</td></tr>
<tr><td colspan="2"></td><td>S</td><td>T</td><td>U</td><td>V</td><td>W</td><td>X</td><td>Y</td><td>Z</td></tr>
<tr><td rowspan="3">Conditions</td><td>Grade C in Computer Science</td><td>Y</td><td>–</td><td>–</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Grade C in Maths</td><td>–</td><td>Y</td><td>Y</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Grade C in Science</td><td>–</td><td>–</td><td>Y</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td rowspan="3">Actions</td><td>Take Computer Science</td><td>Y</td><td>Y</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Take Maths</td><td></td><td>Y</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Take Physics</td><td></td><td></td><td>Y</td><td></td><td></td><td></td><td></td><td></td></tr>
</table></td><td>3</td></tr>
<tr><td>1(c)</td><td>For example:
• (Column S) combining 1,2,3,4…
• … because they only need CS to take CS // Maths and Science do not matter
• (Column T) combining 1,2,5,6…
• … because CS does not matter if it is Y/N
• (Column U) combining 1,5…
• …because CS does not matter if it is Y/N</td><td>3</td></tr>
</table>

Question	Answer	Marks
2(a)	1 mark for each correct line, duration and activity.  	**7**
2(b)	• Dummy activity	**1**

Question	Answer	Marks
3(a)	1 mark per clause • `room(corridor).` • `furniture(table).` • `furniture(lamp).` • `located(table, corridor).` • `located(lamp, corridor).`	**5**
3(b)	• `master_bedroom` • `spare_bedroom`	**2**
3(c)(i)	1 mark per bullet to max 2 • The first clause only says the nursery is next to the master bedroom • … but not that the master bedroom is next to the nursery • The second clause only says the master bedroom is next to the nursery • … but not that the nursery is next to the master bedroom • Goal to find rooms adjacent to master bedroom would not return nursery • … Example. `FindNextTo(X, master_bedroom)` • It is a two-way relationship	**2**
3(c)(ii)	1 mark per bullet • `room(main_bathroom).` • `nextTo(corridor, main_bathroom).` • `nextTo(main_bathroom, corridor).`	**3**

Question	Answer	Marks

3(d)	1 mark per bullet
	• `canBeMovedTo(`**`B,A`**`)`
	• `Furniture(B)`
	• `Room(A)`
	• `AND / ,`
	• `AND NOT / , NOT`
	• `Located(B,A)`
	
	Example:
	
	`canBeMovedTo(B,A)`
	
	`IF furniture(B) AND room(A)`
	
	`AND NOT(located(B,A)).`
		6

Question	Answer	Marks

4(a)	1 mark per item in bold
	
	`FOR Pointer ← 1 TO (Max – 1)`
	
	`  ItemToInsert ← Numbers[Pointer]`
	
	`  CurrentItem ← Pointer`
	
	`  WHILE (CurrentItem > 0) AND (Numbers[CurrentItem – 1] > ItemToInsert)`
	
	`  Numbers[CurrentItem] ← Numbers[CurrentItem – 1]`
	
	`  CurrentItem ← CurrentItem – 1`
	
	`  ENDWHILE`
	
	`  Numbers[CurrentItem] ← ItemToInsert`
	
	`ENDFOR`
		4

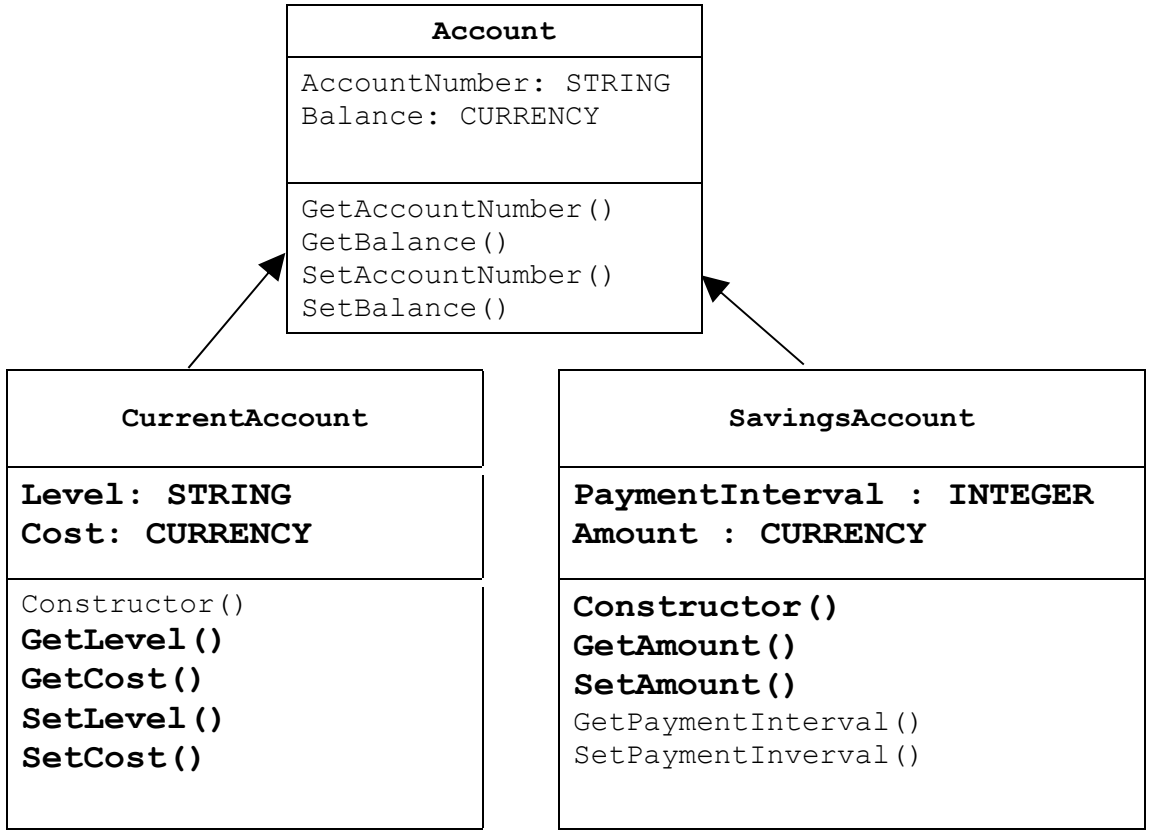
4(b)	• The size of the array // value of Max • How ordered the items already are	**2**

Question	Answer	Marks
5(a)	Max 10	**10**

Label	Op code	Operand	Comment	Marks
START:	LDR	#0	// initialise Index Register	
LOOP:	**LDX**	**LETTERS**	// load LETTERS	1
	CMP	**LETTERTOFIND**	// is LETTERS = LETTERTOFIND ?	1
	JPN	**NOTFOUND**	// if not, go to NOTFOUND	1
	LDD	**FOUND**	// increment FOUND	1
	INC	**ACC**		1
	STO	**FOUND**		1
NOTFOUND:	**LDD**	**COUNT**	//increment COUNT	1
	INC	**ACC**		
	STO	**COUNT**		
	CMP	**#6**	// is COUNT = 6 ?	1
	JPE	**ENDP**	// if yes, end	1
	INC	**IX**	// increment Index Register	1
	JMP	**LOOP**	// go back to beginning of loop	1
ENDP:	END		// end program	
LETTERTOFIND:		'x'		
LETTERS:		'd'		
		'u'		
		'p'		
		'l'		
		'e'		
		'x'		
COUNT:		0		
FOUND:		0		

Question	Answer	Marks
5(b)		**10**

Label	Op Code	Operand		Comment
START:	**LDR**	**#0**	// initialise the Index Register	1
LOOP:	**LDX**	**VALUES**	// load the value from VALUES	1(loop) + 1(LDX Values)
	LSR	**#3**	// divide by 8	1 (LSR) + 1 (#3)
	STX	**VALUES**	// store the new value in VALUES	1
	INC	**IX**	// increment the Index Register	1
	LDD	**REPS**	// increment REPS	1
	INC	**ACC**		
	STO	**REPS**		
	CMP	**#6**	// is REPS = 6 ?	1
	JPN	**LOOP**	// repeat for next value	1
	END			
REPS:	0			
VALUES:	22			
	13			
	5			
	46			
	12			
	33			

Question	Answer	Marks
6(a)	1 mark per bullet • Inheritance correctly shown from `CurrentAccount` and `SavingsAccount` to `Account` • Level and cost methods, get and set functions in `CurrentAccount` • Get and set Amount and constructor in `SavingsAccount`  	**3**

Question	Answer	Marks

6(b)	1 mark per bullet to max 5
	• Class heading and ending
	• Identifying inheritance
	• Declaring `AccountNumber`, `Balance`
	• Use of private/protected for `AccountNumber` and `Balance`
	• One Correct `Get` Method
	• One Correct `Set` Method
	• Second correct `Get` and `Set` Methods
	
	Example VB
	`MustInherit Class Account`
	`  Private AccountNumber As String`
	`  Private Balance As Decimal`
	
	`  Sub SetAccountNumber(AccNumP As String)`
	`  AccountNumber = AccNumP`
	`  End Sub`
	
	`  Function GetAccountNumber() As String`
	`  return AccountNumber`
	`  End Function`
	
	`  Sub SetBalance (BalanceP As Decimal)`
	`  Balance = BalanceP`
	`  End Sub`
	
	`  Function GetBalance() As Decimal`
	`  return Balance`
	`  End Function`
	`End Class`
	
	or
	
	`MustInherit Class Account`
	`  Private AccountNumber As String`
		5

Question	Answer	Marks
6(b)		

```
    Protected AccountNumber As String
       Get
         return _AccountNumber
       End Get
       Set (ByValue AccountNumberV As String)
          _AccountNumber = AccountNumberV
       End Set

    Private _Balance As Decimal
    Protected Balance As Decimal
       Get
         return _Balance
       End Get
       Set (ByValue BalanceV As Integer)
          _Balance = BalanceV
       End Set

 End Class
```

Example Python

```
class Account:
    def__init__(self, accountNumber, balance):
          self.__accountNumber = accountNumber
          self.__balance = balance

    def getAccountNumber(self):
          return self.__accountNumber:
    def setAccountNumber(self, AccountNumber):
        self.___AccountNumber = AcountNumber

    def getBalance(self):
         return self._balance:
    def setBalance(self, Balance):
         self.___Balance = Balance
```

Question	Answer	Marks
6(b)	Example Pascal	

```
type
    Account := class
    private
        AccountNumber, Balance,;
    public
        constructor Create(AccountNumber, Balance);
        procedure setAccountNumber(AccountN: String);
        function getAccountNumber() : String;
        procedure setBalance(BalanceV: Real);
        function getBalance() : Real;

  constructor Account.init(Account, Bal);
  begin
      AccountNumber := Account;
      Balance := Bal;
  end;

  procedure SetAccountNumber(AccountN: String);
  begin
      AccountNumber := AccountN;
  end;

  procedure GetAccountNumber() : String;
  begin
      GetAccountNumber := AccountNumber
  end;

  procedure SetBalance(Bal: String);
  begin
      Balance := Bal;
  end;

  procedure GetBalance() : String;
  begin
```

Question	Answer	Marks

6(b)	`      GetBalance := Balance`
	`  end;`
	`end;`	

6(c)	1 mark per bullet to max 5
	• Class declaration and end
	• Declaration of inheritance
	• `Amount` and `PaymentInterval` as Private/protected with appropriate data types
	
	Constructor:
	• Override / Overriding in constructor
	• Constructor heading and end
	• …taking values as parameters
	• Constructor setting all values using base class
	• Initialisations of new attributes in the constructor
	• … all set to the parameters
	
	Example VB
	`Class SavingsAccount`
	`    Inherits Account`
	`    Private Amount As Decimal`
	`    Private PaymentInterval As Integer`
	
	`    Public Overrides Sub New(ByVal AccountNumberValue As`
	`                  String, ByVal BalanceValue As Decimal, ByVal AmountValue As Decimal,`
	`ByVal PaymentValue As Integer)`
	`        Amount = PaymentValue`
	`        PaymentInterval = PaymentValue`
	`    End Sub`
	
	`End Class`
		5

Question	Answer	Marks

6(c)	or
	
	`Class SavingsAccount`
	`    Inherits Account`
	`    Private Amount As Decimal`
	`    Private PaymentInterval As Integer`
	`    Public Sub New(AccountNumberValue As String, BalanceValue As Decimal, PayInterval As Integer, payAmount As Decimal)`
	`        MyBase.New(AccountNumberValue, BalanceValue)`
	`        AccountNumber = AccountNumberValue`
	`        Balance = BalanceValue`
	`        Amount = payAmount`
	`        PaymentInterval = PayInterval`
	`    End Sub`
	etc.
	
	Example Python
	`class SavingsAccount(Account):`
	
	`    def_init_(self, AccountNumber, Balance, PayInt, AmountP):`
	`        super().__init__(AccountNumber, Balance)`
	`        self.__PaymentInterval = PayInt`
	`        self.__Amount = AmountP`
	
	Example Pascal
	`type`
	`  SavingsAccount = class(Account);`
	`    private`
	`       PaymentInterval : integer;`
	`       Amount : currency;`
	`    public`
	`       constructor Create(AcountNum : String, Bal : Currency, PayInt : Integer, AmountP : Currency);`
	`end;`
	
	`constructor SavingsAccount.Create(); override;`	

Question	Answer	Marks

6(c)	`begin`
	`    inherited Create(AccountNum, Bal)`
	`    PaymentInterval := PayInt;`
	`    Amount := AmountP;`
	`end;`	