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 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 **20** printed pages.

Cambridge Assessment International Education

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:

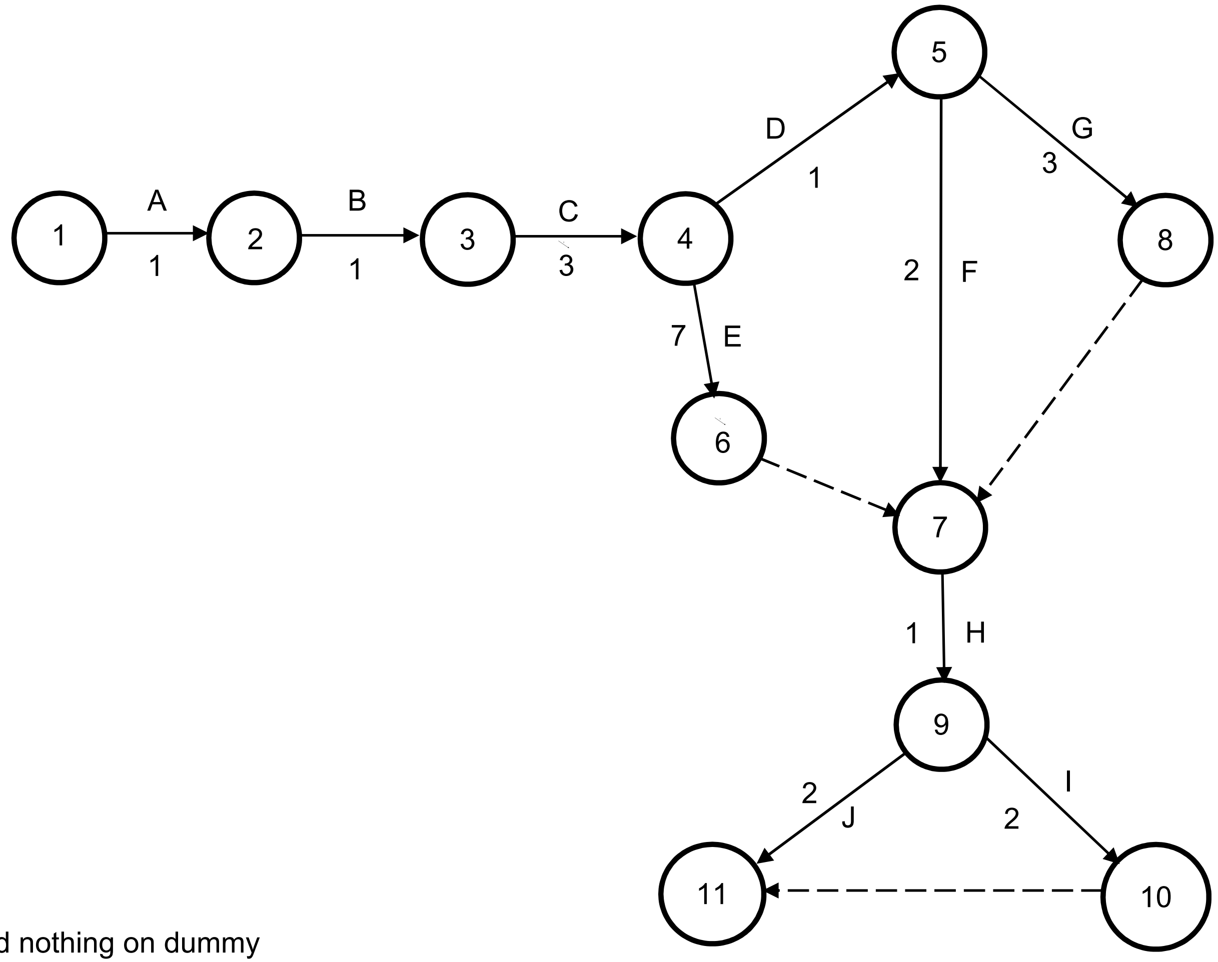
Marks must be awarded **positively**:

- 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.

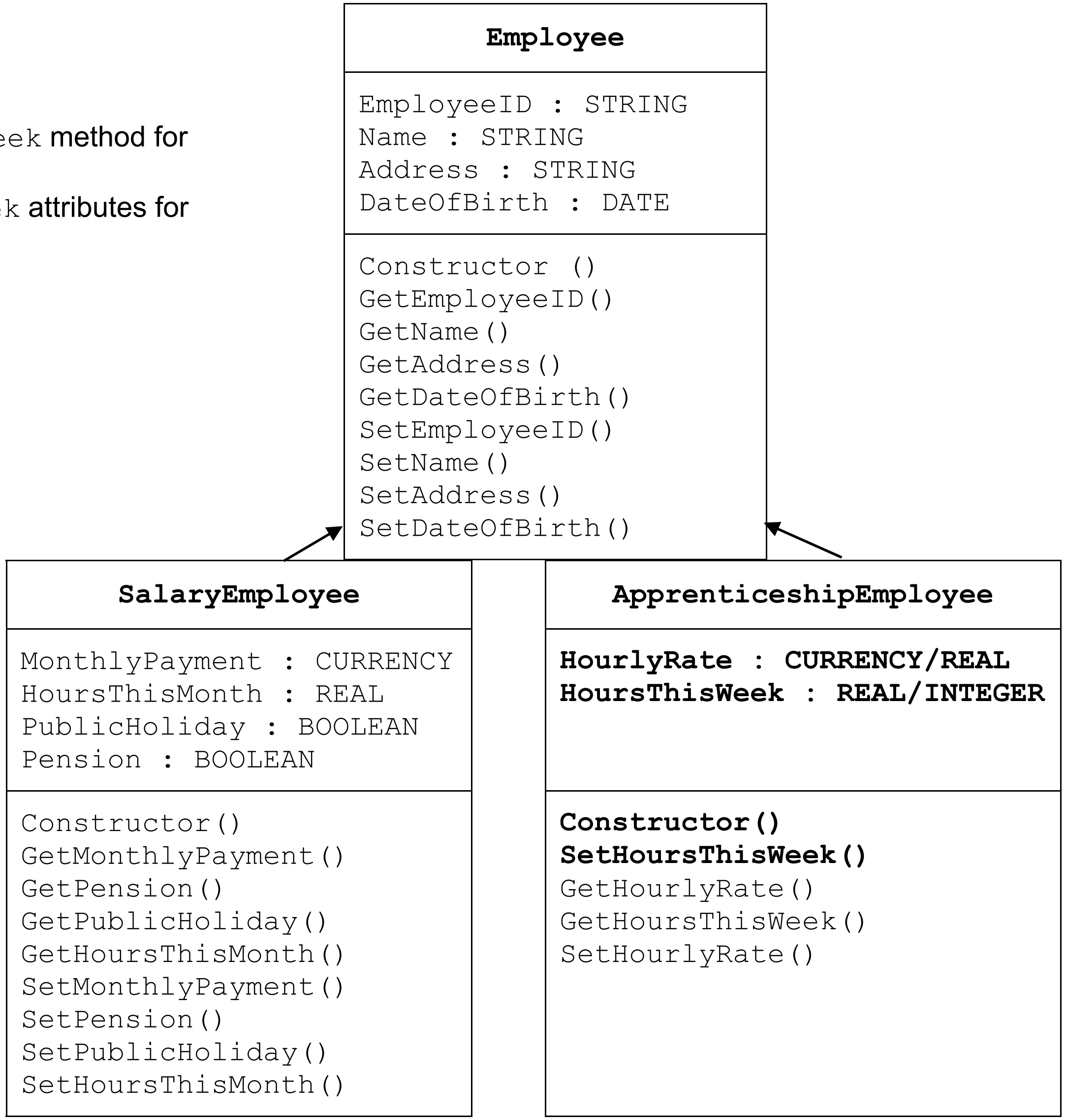
GENERIC MARKING PRINCIPLE 5: Marks should be awarded using the full range of marks defined in the mark scheme for the question (however; the use of the full mark range may be limited according to the quality of the candidate responses seen).
GENERIC MARKING PRINCIPLE 6: Marks awarded are based solely 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 2 3 4 5 6 7 8 9 10 11 A 1 B 1 C 3 D 1 E 7 F 2 G 3 H 1 I 2 J 2 1 mark each bullet point: • B 1 • C 3 (following B) • D 10 (following C) • G 3 (following D) and nothing on dummy • E 7 (following C) and nothing on dummy • H1 in position • J 2 (following H) and nothing on dummy	**7**
1(a)(ii)	1 mark: • The next activity is dependent on the previous but there is no activity	**1**

Question	Answer	Marks
1(b)	1 mark per row	**3**

		Rules							
Conditions	**Public Holiday**	Y	Y	Y	Y	N	N	N	N
	Hours >= 160	Y	Y	N	N	Y	Y	N	N
	Pension	Y	N	Y	N	Y	N	Y	N
Actions	**3% bonus payment**	**X**	**X**	**X**	**X**				
	5% bonus payment	**X**	**X**			**X**	**X**		
	4% Pension payment	**X**		**X**		**X**		**X**	

Question	Answer	Marks

1(c)(i)	1 mark per bullet point
	• inheritance (arrow filled or unfilled)
	• constructor and `SetHoursThisWeek` method for `ApprenticeshipEmployee`
	• `HourlyRate` and `HoursThisWeek` attributes for `ApprenticeshipEmployee`
	**Employee**
	EmployeeID : STRING
	Name : STRING
	Address : STRING
	DateOfBirth : DATE
	Constructor ()
	GetEmployeeID()
	GetName()
	GetAddress()
	GetDateOfBirth()
	SetEmployeeID()
	SetName()
	SetAddress()
	SetDateOfBirth()
	SalaryEmployee
	MonthlyPayment : CURRENCY
	HoursThisMonth : REAL
	PublicHoliday : BOOLEAN
	Pension : BOOLEAN
	Constructor()
	GetMonthlyPayment()
	GetPension()
	GetPublicHoliday()
	GetHoursThisMonth()
	SetMonthlyPayment()
	SetPension()
	SetPublicHoliday()
	SetHoursThisMonth()
	ApprenticeshipEmployee
	HourlyRate : CURRENCY/REAL
	HoursThisWeek : REAL/INTEGER
	Constructor()
	SetHoursThisWeek()
	GetHourlyRate()
	GetHoursThisWeek()
	SetHourlyRate()
		3

Question	Answer	Marks
1(c)(ii)	1 mark per bullet point	4

- Constructor header and close (where necessary)
- All 4 values sent as parameters (ID, Name, Address, DateOfBirth) with any
- Attributes/properties set to a value …
- … that are the parameters

Example code:

Pascal

```
Constructor Employee.init(ID, NewName, NewAddress, NewDateOfBirth : String);
begin
  EmployeeID := ID;
  Name := NewName;
  Address := NewAddress;
  DateOfBirth := NewDateOfBirth;
end;
```

Python

```
def __init__(self, ID, NewName, NewAddress, NewDateOfBirth):
  self.__EmployeeID = ID
  self.__Name = NewName
  self.__Address = NewAddress
  self.__DateOfBirth = NewDateOfBirth
```

VB.NET

```
Public Sub New(ID, NewName, NewAddress, NewDateOfBirth)
  EmployeeID = ID
  Name = NewName
  Address = NewAddress
  DateOfBirth = NewDateOfBirth
End Sub
```

Question	Answer	Marks

1(c)(iii)	1 mark per bullet point
	• Get method header and close (where needed) with no parameters
	• Returns the attribute/property `EmployeeID`
	Example code:
	Pascal
	`Function  Employee.GetEmployeeID() : String;`
	`  result := EmployeeID;`
	`end;`
	Python
	`def GetEmployeeID(self):`
	`  return self.__EmployeeID`
	VB.NET
	`Public Function GetEmployeeID() As String`
	`  Return EmployeeID`
	`End Function`
		2

Question	Answer	Marks

1(c)(iv)	1 mark per bullet point
	• Set method/procedure header and close (where needed) **with parameter**
	• Sets `EmployeeID` to value of parameter
	Example code:
	Pascal
	`procedure Employee.SetEmployeeID(NewID: String);`
	`  EmployeeID := NewID`
	`end;`
	Python
	`def SetEmployeeID(self, NewID):`
	`  self.__EmployeeID = NewID`
	VB.NET
	`Public Sub SetEmployeeID(NewID)`
	`  EmployeeID = NewID`
	`End Sub`
		2

Question	Answer	Marks
1(c)(v)	1 mark per bullet point	4

- Set method/function header and close (where needed) with parameter
- Checking value of parameter for both true and false …
- … If valid – setting Pension to parameter **and** returning True
- … If not valid – **not** setting Pension **and** returning False

Example code:

Pascal

```
Function salaryEmployee.SetPension(NewPension) : boolean;
  IF NewPension = true or NewPension = false THEN
    Pension := NewPension;
    Result := true;
  ELSE
    Result := false;
end;
```

Python

```
def SetPension(self, NewPension):
  if NewPension == True Or NewPension == False:
    self.__Pension = NewPension
    return True
  else:
    return False
```

VB.NET

```
Public Function SetPension(NewPension) AS Boolean
  If NewPension = True Or NewPension = False Then
    Pension = NewPension
    Return True
  Else
    Return False
  End If
End Function
```

Question	Answer	Marks
1(c)(vi)	1 mark per bullet point to **max 8**	**8**

- Function header and close (where needed) with at least **one** parameter
- Constants used for hours bonus, month bonus, holiday bonus, pension cost(at least 3)
- Checking if Hours is > = 160, calculating bonus payment (monthlypay * 0.05)
- Checking if pension is true, calculation pension to pay (monthlypay * 0.04)
- Checking if public holiday is true, calculation of bonus payment (monthlypay * 0.03)
- … all 3 Hours, Pension and `PublicHoliday` accessed from the parameter using Get methods

- Adding both bonus payments to basic salary and deducting pension from salary
- … basic salary accessed by using `GetMonthlyPayment` with the parameter
- Outputting the **total** final bonus and pension with appropriate message
- Returning the new salary

Example code:

Pascal

```
Function CalculateMonthlySalary(TheEmployee : SalaryEmployee) : real;
  var
    BonusPayment : real;
    PensionPayment : real;
    BasicSalary : real;

  const
   HoursBonus : real = 0.05;
   HoursMonthBonus : integer = 160;
   PensionCost : real = 0.04;
   PublicHolidayBonus : real = 0.03;
  begin
    BonusPayment :=0;
    PensionPayment :=0;
    BasicSalary :=0;
```

Question	Answer	Marks
1(c)(vi)		

```
    BasicSalary := TheEmployee.GetMonthlyPayment();

    IF TheEmployee.GetHoursThisMonth() >= HoursMonthBonus THEN
      BonusPayment := BasicSalary * HoursBonus;

    IF TheEmployee.GetPension() = True THEN
      PensionPayment := BasicSalary * PensionCost;

    IF TheEmployee.GetPublicHoliday() = True THEN
      BonusPayment := BonusPayment + BasicSalary * PublicHolidayBonus;


    writeln("The pension payment is " & PensionPayment);
    writeln ("The bonus payment is " & BonusPayment);

    MonthlySalary := BasicSalary + BonusPayment - PensionPayment;
    result :=  MonthlySalary;
  end;
```

Python

```
def CalculateMonthlySalary(TheEmployee):

  BonusPayment = 0
  PensionPayment = 0
  HoursBonus = 0.05
  HoursMonthBonus = 160
  PensionCost = 0.04
  PublicHolidayBonus = 0.03

  BasicSalary = TheEmployee.GetMonthlyPayment()

  if TheEmployee.GetHoursThisMonth() >= HoursMonthBonus:
    BonusPayment = BasicSalary * HoursBonus
```

Question	Answer	Marks
1(c)(vi)		

```
  if TheEmployee.GetPension() == true:
    PensionPayment = BasicSalary * PensionCost

  if TheEmployee.GetPublicHoliday() == true:
    BonusPayment = BonusPayment + BasicSalary * PublicHolidayBonus

  print("The pension payment is ",  str(PensionPayment))
  print("The bonus payment is ", str(BonusPayment))

  MonthlySalary = BasicSalary + BonusPayment - PensionPayment
  return MonthlySalary
```

VB.NET

```
Public Function CalculateMonthlySalary(TheEmployee As SalaryEmployee) As Double

  Dim BonusPayment As Single = 0
  Dim PensionPayment As Single = 0
  Dim MonthlySalary As Single = 0

  Const HoursBonus As Single = 0.05
  Const HoursMonthBonus As Integer = 160
  Const PensionCost As Single = 0.04
  Const PublicHolidayBonus As Single = 0.03

  Dim BasicSalary As Double = TheEmployee.GetMonthlyPayment()

  If TheEmployee.GetHoursThisMonth() >= HoursMonthBonus Then
    BonusPayment = BasicSalary * HoursBonus
  End If
  If TheEmployee.GetPension() = True Then
    PensionPayment = BasicSalary * PensionCost
  End If
  If TheEmployee.GetPublicHoliday() = True Then
    BonusPayment = BonusPayment + BasicSalary * PublicHolidayBonus
  End If
```

Question	Answer	Marks

1(c)(vi)	`  Console.WriteLine("The pension payment is " & PensionPayment)`
	`  Console.WriteLine("The bonus payment is " & BonusPayment)`
	
	`  MonthlySalary = BasicSalary + BonusPayment - PensionPayment`
	`  Return MonthlySalary`
	`End Function`	

1(d)	Polymorphism	**1**

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>2(a)</td><td>

1 mark per completed statement

```
FUNCTION AddToQueue(Number : INTEGER) RETURNS BOOLEAN
  CONSTANT FirstIndex = 0
  CONSTANT LastIndex = 7
  TempPointer ← EndPointer + 1
  IF TempPointer > LastIndex
   THEN
    TempPointer ← FirstIndex
  ENDIF
  IF TempPointer = StartPointer
   THEN
     RETURN FALSE
   ELSE
    EndPointer ← TempPointer
    NumberQueue[EndPointer] ← Number
    RETURN TRUE
  ENDIF
ENDFUNCTION
```

</td><td>5</td></tr>
<tr><td>2(b)</td><td>

1 mark per bullet point

1 mark for:

- … if the start pointer reaches the end of the queue, it becomes the index of the first element in the queue

Max 3 from:

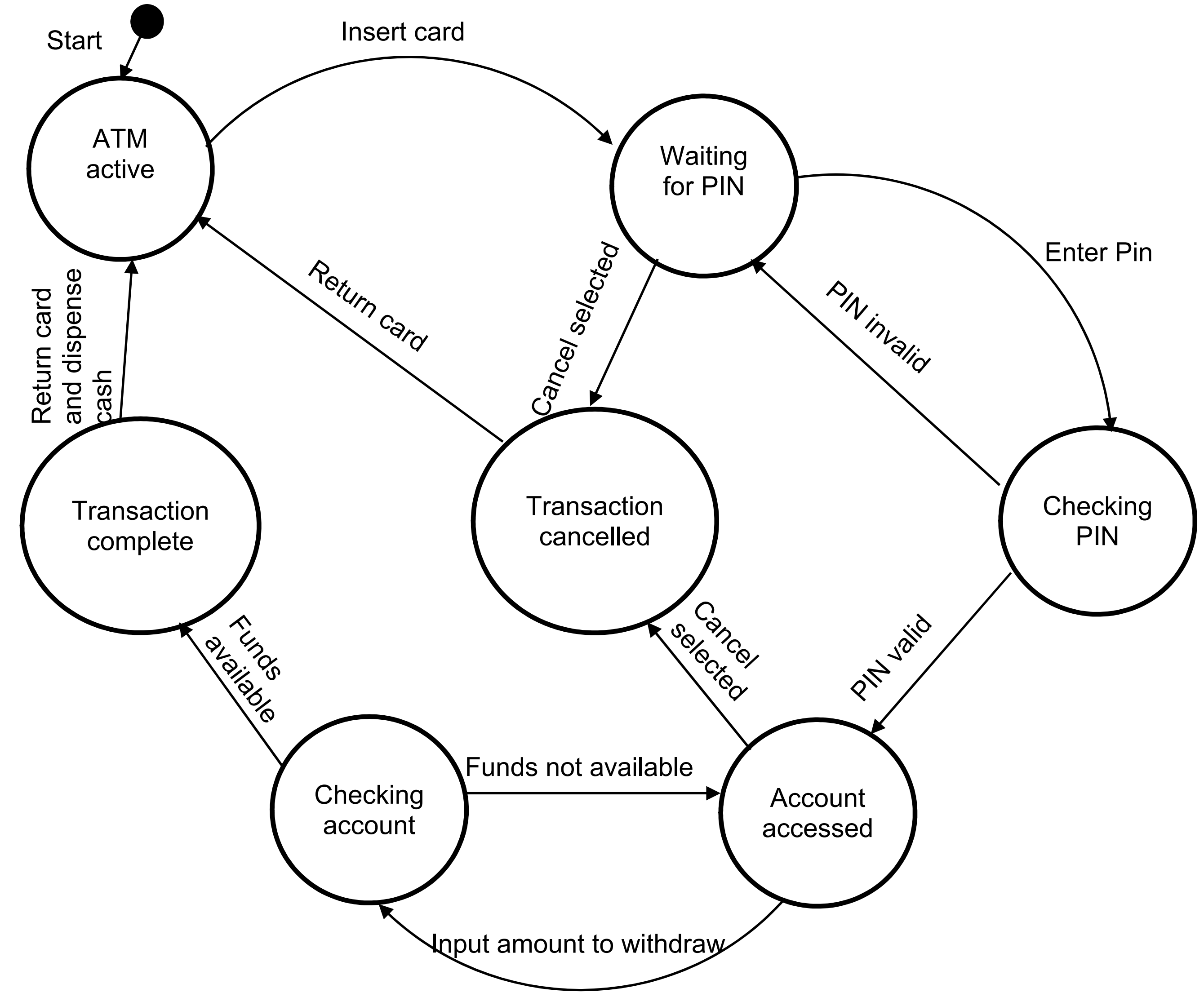
- Checks if the circular queue is empty // Checks if the queue has any data in it
- … if it is empty it **reports** that it is empty
- If not empty, return the value at the position of the **start pointer** …
- … then increments the start pointer

</td><td>4</td></tr>
</table>

Question	Answer	Marks
2(c)	1 mark per bullet point to **max 3** e.g. • Stack • Linked list • Dictionary • (Binary) tree	**3**

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>3(a)</td><td>1 mark per test data type
• Normal / valid
• Abnormal / erroneous / invalid
• Boundary / extreme</td><td><b>3</b></td></tr>
<tr><td>3(b)(i)</td><td>1 mark for each name
<table>
<tr><th>Description</th><th>Name of debugging feature</th></tr>
<tr><td>A point where the program can be halted to see if the program works to this point</td><td><b>Breakpoint</b></td></tr>
<tr><td>One statement is executed and then the program waits for input from the programmer to move to the next statement.</td><td><b>Stepping // step through/over/into</b></td></tr>
</table></td><td><b>2</b></td></tr>
<tr><td>3(b)(ii)</td><td>1 mark for name, 1 for description
e.g.
• variable watch window
• observe how variables change during execution // view current status of variables

• Error list
• describes the error // gives line number of error</td><td><b>2</b></td></tr>
</table>

Question	Answer	Marks
4	1 mark for each correct transition with arrow.  Start Insert card ATM active Waiting for PIN Enter Pin Return card Cancel selected PIN invalid Return card and dispense cash Transaction complete Transaction cancelled Checking PIN Funds available Cancel selected PIN valid Checking account Funds not available Account accessed Input amount to withdraw	8

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>5(a)</td><td>1 mark for each shaded section
• LDD NUMBER
• LSL
• #2
• STO NUMBER
• END
<table>
<tr><th>Label</th><th>Op code</th><th>Operand</th><th>Comment</th></tr>
<tr><td></td><td><b>LDD</b></td><td><b>NUMBER</b></td><td>// load contents of NUMBER</td></tr>
<tr><td></td><td><b>LSL</b></td><td><b>#2</b></td><td>// perform shift to multiply by 4</td></tr>
<tr><td></td><td><b>STO</b></td><td><b>NUMBER</b></td><td>// store contents of ACC in NUMBER</td></tr>
<tr><td></td><td><b>END</b></td><td></td><td>// end program</td></tr>
<tr><td>NUMBER:</td><td colspan="2">B00110110</td><td></td></tr>
</table>
</td><td><b>5</b></td></tr>
</table>

Question	Answer					Marks
5(b)						**8**
	Label	**Op code**	**Operand**	**Comment**		
		LDR	#0	// initialise index register to 0		
	START:	**LDX**	**STRING**	// load the next value from STRING	**1**	
		AND	**MASK**	// bitwise AND operation with MASK	**1**	
		CMP	**MASK**	// check if result equals MASK	**1**	
		JPN	**UPPER**	// if FALSE jump to UPPER	**1**	
		LDD	**COUNT**	// increment COUNT	**1**	
		INC	**ACC**			
		STO	**COUNT**			
	UPPER:	INC	IX	// increment Index Register	**1**	
		LDD	**LENGTH**	// decrement LENGTH	**1**	
		DEC	**ACC**			
		STO	**LENGTH**			
		CMP	**#0**	// is LENGTH = 0 ?	**1**	
		JPN	**START**	// if FALSE, jump to START	**1**	
		END		// end program		

Question	Answer				Marks
5(b)	MASK:	B00100000	// if bit 5 is 1, letter is lower case		
	COUNT:	0			
	LENGTH:	5			
	STRING:	B01001000	// The ASCII code for 'H'		
		B01100001	// The ASCII code for 'a'		
		B01110000	// The ASCII code for 'p'		
		B01110000	// The ASCII code for 'p'		
		B01011001	// The ASCII code for 'Y'		