Example Candidate Responses

Cambridge International AS & A Level

Cambridge International AS & A Level Computer Science

9608

Paper 4

Cambridge Advanced

CAMBRIDGE International Examinations

Version 1

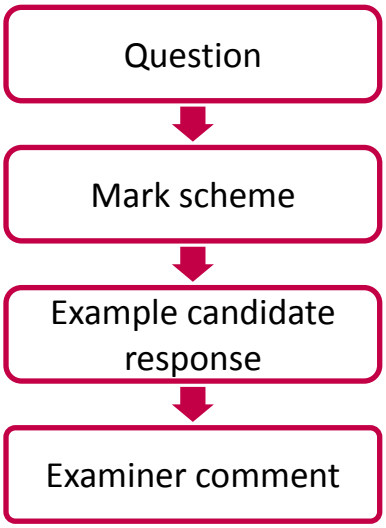
Contents

Introduction

The main aim of this booklet is to exemplify standards for those teaching Cambridge International AS & A Level Computer Science (9608), and to show how different levels of candidates' performance relate to the subject's curriculum and assessment objectives.

In this booklet candidate responses have been chosen to exemplify a range of answers. Each response is accompanied by a brief commentary explaining the strengths and weaknesses of the answers.

For ease of reference the following format for each component has been adopted:



Each question is followed by an extract of the mark scheme used by examiners. This, in turn, is followed by examples of marked candidate responses, each with an examiner comment on performance. Comments are given to indicate where and why marks were awarded, and how additional marks could have been obtained. In this way, it is possible to understand what candidates have done to gain their marks and what they still have to do to improve their marks.

This document illustrates the standard of candidate work for those parts of the assessment which help teachers assess what is required to achieve marks beyond what should be clear from the mark scheme. Some question types where the answer is clear from the mark scheme, such as short answers and multiple choice, have therefore been omitted.

Past papers, Examiner Reports and other teacher support materials are available on Teacher Support at https://teachers.cie.org.uk

Assessment at a glance

For Cambridge International AS and A Level Computer Science, candidates may choose:

- to take Papers 1, 2, 3 and 4 in the same examination series, leading to the full Cambridge International A Level
- to follow a **staged** assessment route by taking Papers 1 and 2 (for the AS Level qualification) in one series, then Papers 3 and 4 (for the full Cambridge International A Level) in a later series
- to take Papers 1 and 2 only (for the AS Level qualification).

Components All candidates take	Weighting (%)	
	AS	A
Paper 1 Theory Fundamentals This written paper contains short-answer and structured questions. There is no choice of questions. 75 marks Externally assessed 1 hour 30 minutes	50	25
Paper 2 Fundamental Problem-solving and Programming Skills This written paper contains short-answer and structured questions. There is no choice of questions. Topics will include those given in the pre-release material.[1] 75 marks Externally assessed 2 hours	50	25
Paper 3 Advanced Theory This written paper contains short-answer and structured questions. There is no choice of questions. 75 marks Externally assessed 1 hour 30 minutes	–	25
Paper 4 Further Problem-solving and Programming Skills This written paper contains short-answer and structured questions. There is no choice of questions. Topics will include those given in the pre-release material.[1] 75 marks Externally assessed 2 hours	–	25

Advanced Subsidiary (AS) forms 50% of the assessment weighting of the full Advanced (A) Level.

Teachers are reminded that the latest syllabus is available on our public website at **www.cie.org.uk** and Teacher Support at **https://teachers.cie.org.uk**

Paper 4 – Further Problem-solving and Programming Skills

Question 1

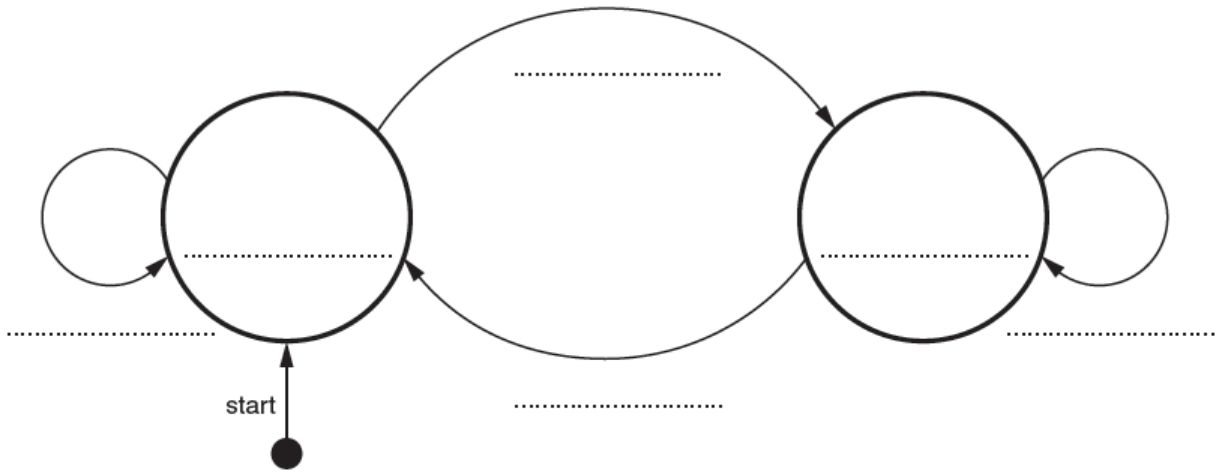
1 A turnstile is a gate which is in a locked state. To open it and pass through, a customer inserts a coin into a slot on the turnstile. The turnstile then unlocks and allows the customer to push the turnstile and pass through the gate.

After the customer has passed through, the turnstile locks again. If a customer pushes the turnstile while it is in the locked state, it will remain locked until another coin is inserted.

The turnstile has two possible states: **locked** and **unlocked**. The transition from one state to another is as shown in the table below.

Current state	Event	Next state
Locked	Insert coin	Unlocked
Locked	Push	Locked
Unlocked	Attempt to insert coin	Unlocked
Unlocked	Pass through	Locked

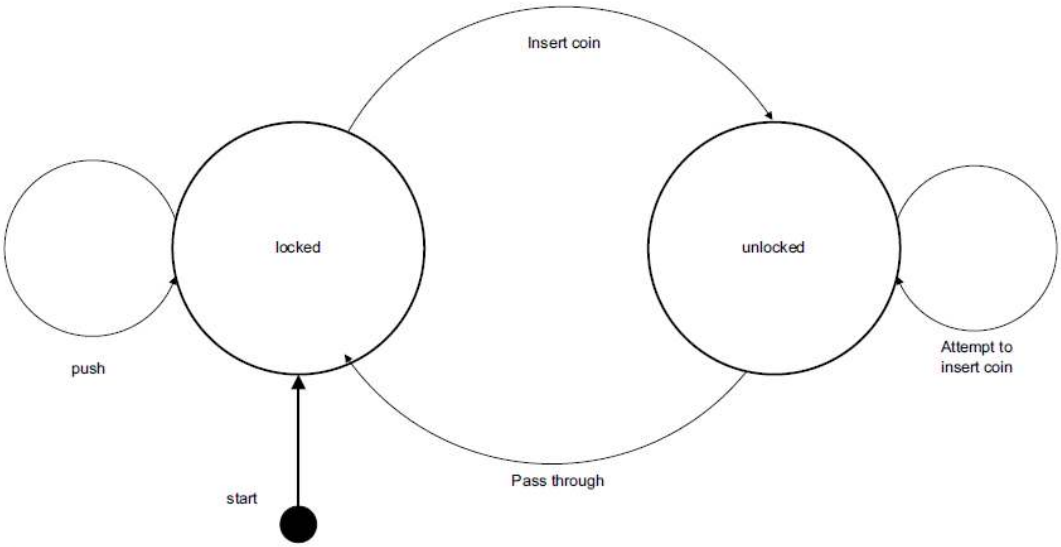
Complete the state transition diagram for the turnstile:



[5]

Mark scheme

1



Mark as follows:
1 mark for both states correct
1 mark for each further label

[5]

Example candidate response – high

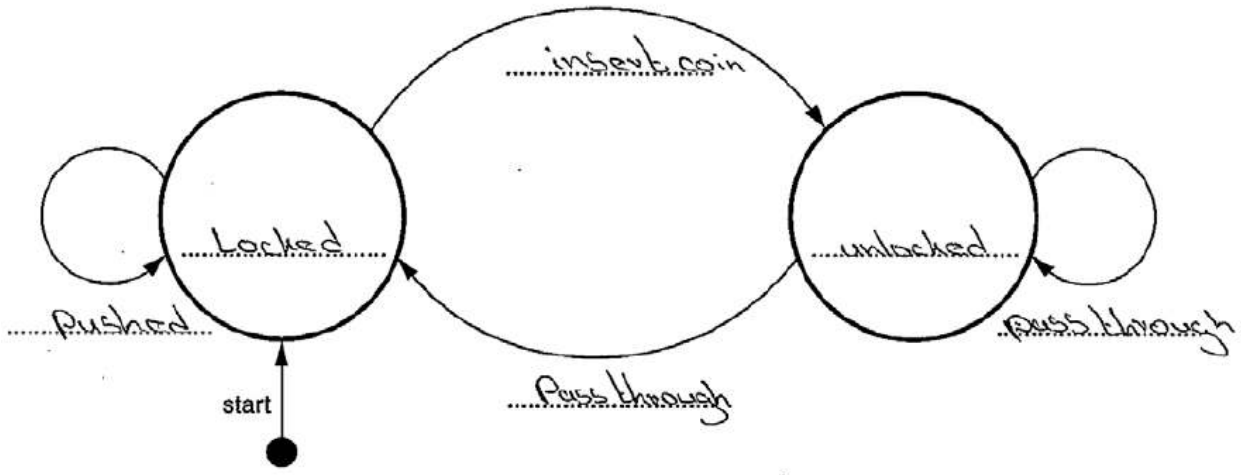
1 A turnstile is a gate which is in a locked state. To open it and pass through, a customer inserts a coin into a slot on the turnstile. The turnstile then unlocks and allows the customer to push the turnstile and pass through the gate.

After the customer has passed through, the turnstile locks again. If a customer pushes the turnstile while it is in the locked state, it will remain locked until another coin is inserted.

The turnstile has two possible states: **locked** and **unlocked**. The transition from one state to another is as shown in the table below.

Current state	Event	Next state
Locked	Insert coin	Unlocked
Locked	Push	Locked
Unlocked	Attempt to insert coin	Unlocked
Unlocked	Pass through	Locked

Complete the state transition diagram for the turnstile:



[5]

Examiner comment – high

Here the candidate correctly labelled the states. The possible events when the turnstile is in the locked state are correctly identified. The candidate did not appreciate that from the unlocked state there can't be the same event resulting in two different states. The event 'attempt to insert coin' was not identified.

Total marks awarded = 4 out of 5

Example candidate response – middle

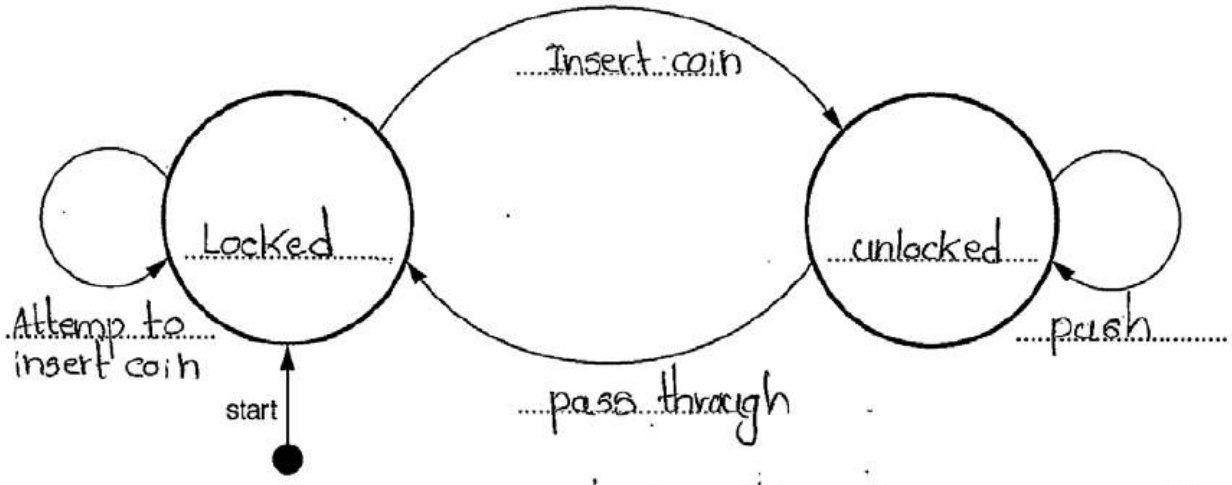
1 A turnstile is a gate which is in a locked state. To open it and pass through, a customer inserts a coin into a slot on the turnstile. The turnstile then unlocks and allows the customer to push the turnstile and pass through the gate.

After the customer has passed through, the turnstile locks again. If a customer pushes the turnstile while it is in the locked state, it will remain locked until another coin is inserted.

The turnstile has two possible states: **locked** and **unlocked.** The transition from one state to another is as shown in the table below.

Current state	Event	Next state
Locked	Insert coin	Unlocked
Locked	Push	Locked
Unlocked	Attempt to insert coin	Unlocked
Unlocked	Pass through	Locked

Complete the state transition diagram for the turnstile:



[5]

Examiner comment – middle

The candidate correctly labelled the states and the events that result in a different state. The events that do not change the current state were applied to the wrong states.

Total marks awarded = 3 out of 5

Example candidate response – low

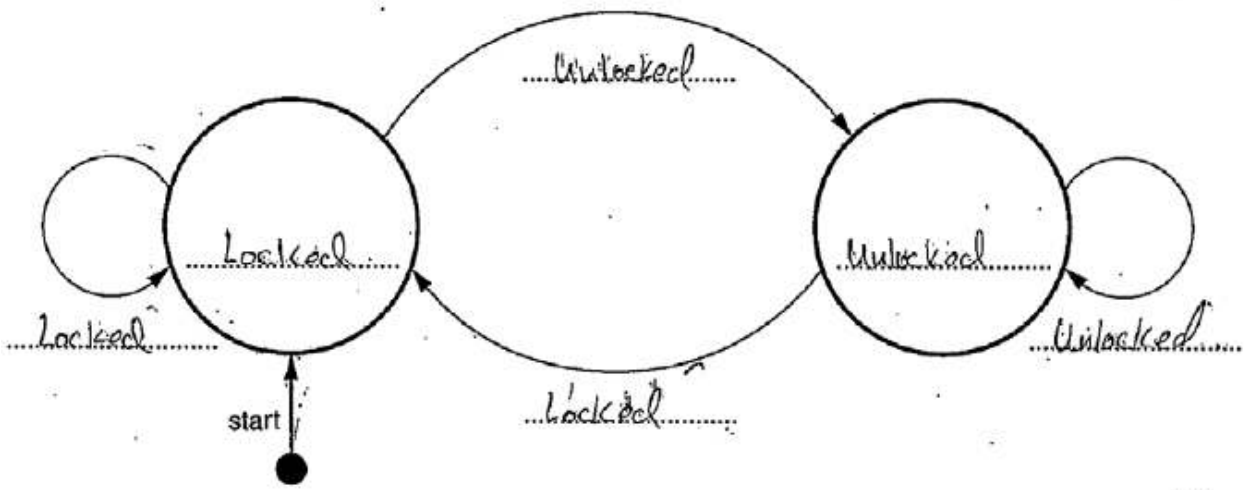
1 A turnstile is a gate which is in a locked state. To open it and pass through, a customer inserts a coin into a slot on the turnstile. The turnstile then unlocks and allows the customer to push the turnstile and pass through the gate.

After the customer has passed through, the turnstile locks again. If a customer pushes the turnstile while it is in the locked state, it will remain locked until another coin is inserted.

The turnstile has two possible states: **locked** and **unlocked**. The transition from one state to another is as shown in the table below.

Current state	Event	Next state
Locked	Insert coin	Unlocked
Locked	Push	Locked
Unlocked	Attempt to insert coin	Unlocked
Unlocked	Pass through	Locked

Complete the state transition diagram for the turnstile:



[5]

Examiner comment – low

The candidate correctly labelled the states, noting that the turnstile starts in the locked state, as given in the introduction of the question. However, the candidate did not appreciate that the arrows in the diagram represent events.

Total marks awarded = 1 out of 5

Question 2

2 A declarative programming language is used to represent the knowledge base shown below:

```
01  capital_city(amman).
02  capital_city(beijing).
03  capital_city(brussels).
04  capital_city(cairo).
05  capital_city(london).
06  city_in_country(amman, jordan).
07  city_in_country(shanghai, china).
08  city_in_country(brussels, belgium).
09  city_in_country(london, uk).
10  city_in_country(manchester, uk).
11  country_in_continent(belgium, europe).
12  country_in_continent(china, asia).
13  country_in_continent(uk, europe).
14  city_visited(amman).
15  city_visited(beijing).
16  city_visited(cairo).
```

These clauses have the following meaning:

Clause	Explanation
01	Amman is a capital city
06	Amman is a city in the country of Jordan
11	Belgium is a country in the continent of Europe
14	The travel writer visited Amman

(a) More facts are to be included.

The travel writer visited the city of Santiago which is the capital city of Chile, in the continent of South America.

Write additional clauses to record this.

17 .....

.....

18 .....

.....

19 .....

.....

20 .....

..... [4]

Question 2, continued

(b) Using the variable `ThisCountry`, the goal

```
country_in_continent(ThisCountry, europe)
```

returns

```
ThisCountry = belgium, uk
```

Write the result returned by the goal:

```
city_in_country(ThisCity, uk)
```

`ThisCity =` .....

..... [2]

(c) Complete the rule below to list the countries the travel writer has visited.

```
countries_visited(ThisCountry)
```

`IF` .....

.....

.....

.....

..... [4]

Mark scheme

2 (a)

```
capital_city(santiago).
city_in_country(santiago, chile).
country_in_continent(chile,south_america).
city_visited(santiago).
```

accept in any order **[4]**

(b)

```
ThisCity =
manchester
london
```

[2]

(c)

```
countries_visited(ThisCountry)
IF
city_visited(ThisCity)                          1
AND                                             1
city_in_country(ThisCity, ThisCountry)          2
```

[4]

Example candidate response – high

(a) More facts are to be included.

The travel writer visited the city of Santiago which is the capital city of Chile, in the continent of South America.

Write additional clauses to record this.

17 capital_city(santiago)

18 city_in_country(santiago, chile)

19 country_in_continent(chile, southamerica)

20 city_visited(santiago)

[4]

(b) Using the variable `ThisCountry`, the goal

```
country_in_continent(ThisCountry, europe)
```

returns

```
ThisCountry = belgium, uk
```

Write the result returned by the goal:

```
city_in_country(ThisCity, uk)
```

ThisCity = london, manchester

[2]

(c) Complete the rule below to list the countries the travel writer has visited.

```
countries_visited(ThisCountry)
```

IF city_visited(ThisCity) = city_in_country(ThisCountry, ThisCity)
then output(ThisCountry)

returns

forall ThisCountry = Jordan, China, Egypt, Chile

[4]

Examiner comment – high

In part (a) the candidate converted the given facts correctly into clauses, taking care to show that atoms and predicates are written with a lower case first letter.

In part (b) the candidate knew that the variable 'ThisCity' would instantiate first to London and then to Manchester.

In part (c) the candidate realised that the clauses 'city_visited' and 'city_in_country' are needed to find out which countries the travel writer has visited. The candidate also realised that variables were required. However, the variables for 'city_in_country' were not in the correct order as the second variable represents the country and therefore needs to match the variable used in the head of the rule. The response then continues as an imperative IF statement rather than the required declarative statement making a rule.

Marks awarded for part (a) = 4/4
Marks awarded for part (b) = 2/2
Marks awarded for part (c) = 2/4

Total marks awarded = 8 out of 10

Example candidate response – middle

(a) More facts are to be included.

The travel writer visited the city of Santiago which is the capital city of Chile, in the continent of South America.

Write additional clauses to record this.

17 capital_city (Santiago).

18 city_in_country (Santiago, Chile).

19 country_in_continent (chile, south_america).

20 city_visited (santiago).

[4]

(b) Using the variable `ThisCountry`, the goal

```
country_in_continent(ThisCountry, europe)
```

returns

```
ThisCountry = belgium, uk
```

Write the result returned by the goal:

```
city_in_country(ThisCity, uk)
```

ThisCity = london, manchester

[2]

(c) Complete the rule below to list the countries the travel writer has visited.

```
countries_visited(ThisCountry)
IF city_visited(ThisCity) is not nothing
THEN
    city_in_country(ThisCity, ThisCountry)
    countries_visited(ThisCountry)
ENDIF
```

[4]

Examiner comment – middle

In part (a) the candidate converted the given facts correctly into clauses. However, the response does not clearly show that atoms and predicates are written with a lower case first letter.

In part (b) the candidate knew that the variable 'ThisCity' would instantiate first to London and then to Manchester.

In part (c) the candidate realised that the clauses 'city_visited' and 'city_in_country' are needed to find out which countries the travel writer has visited. The candidate also realised that variables were required. However, the response is written as an imperative IF statement rather than the required declarative statement representing a rule.

Marks awarded for part (a) = 3/4
Marks awarded for part (b) = 2/2
Marks awarded for part (c) = 1/4

Total marks awarded = 6 out of 10

Example candidate response – low

(a) More facts are to be included.

The travel writer visited the city of Santiago which is the capital city of Chile, in the continent of South America.

Write additional clauses to record this.

17 capital_city(Santiago)

18 city_in_country(Santiago, Chile)

19 country_in_continent(Chile, South America)

20 city_visited(Santiago)

[4]

(b) Using the variable `ThisCountry`, the goal

```
country_in_continent(ThisCountry, europe)
```

returns

```
ThisCountry = belgium, uk
```

Write the result returned by the goal:

```
city_in_country(ThisCity, uk)
```

`ThisCity` = london, uk

[2]

(c) Complete the rule below to list the countries the travel writer has visited.

```
countries_visited(ThisCountry)
IF city_visited(Cammun) then
    countries_visited(jordan)
else if city_visited(Santiago) then
    countries_visited(Chile)
End if
```

[4]

Candidate script = 654551214
(file name ***or*** centre & candidate number ***or*** Scoris/Assessor ID)

Examiner comment – low

In part (a) the candidate converted the given facts correctly into clauses. However, the response clearly shows the atoms written with an upper case first letter.

In part (b) the candidate correctly stated that the variable 'ThisCity' would instantiate to London. The candidate did not appear to be aware that the variable could instantiate to other atoms when testing the remainder of the knowledge base.

In part (c) the candidate responded with an imperative IF statement using some of the facts in the knowledge base, but this is not a rule for a declarative program.

Marks awarded for part (a) = 3/4
Marks awarded for part (b) = 1/2
Marks awarded for part (c) = 0/4

Total marks awarded = 4 out of 10

Question 3

3 A shop gives some customers a discount on goods totalling more than $20.
The discounts are:

- 5% for goods totalling more than $100
- 5% with a discount card
- 10% with a discount card and goods totalling more than $100

(a) Complete the decision table.

Conditions	goods totalling more than $20	Y	Y	Y	Y	N	N	N	N
	goods totalling more than $100	Y	Y	N	N	Y	Y	N	N
	have discount card	Y	N	Y	N	Y	N	Y	N
Actions	No discount								
	5% discount								
	10% discount								

[4]

(b) Simplify your solution by removing redundancies.

Conditions	goods totalling more than $20								
	goods totalling more than $100								
	have discount card								
Actions	No discount								
	5% discount								
	10% discount								

[5]

Question 3, continued

(c) The simplified table produced in **part (b)** is used as a design for program code.

The following identifier table shows the parameters to be passed to the function `Discount`. This function returns the discount amount as an integer.

Identifier	Data type
GoodsTotal	INTEGER
HasDiscountCard	BOOLEAN

Write **program code** for this function.

Programming language .....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [6]

Mark scheme

3 (a)

Conditions	goods totalling more than $20	Y	Y	Y	Y	N	N	N	N
	goods totalling more than $100	Y	Y	N	N	Y	Y	N	N
	have discount card	Y	N	Y	N	Y	N	Y	N
Actions	No discount				X	X	X	X	X
	5% discount		X	X					
	10% discount	X							
		1 mark	1 mark	1 mark	1 mark				

[4]

(b)

Conditions	goods totalling more than $20	Y	Y	Y	Y	N			
	goods totalling more than $100	Y	Y	N	N	-			
	have discount card	Y	N	Y	N	-			
Actions	No discount				X	X			
	5% discount		X	X					
	10% discount	X							

1 mark per column [5]

Mark scheme, continued

(c) Example Pascal

```
      FUNCTION Discount(GoodsTotal: INTEGER; HasDiscountCard: BOOLEAN) :
INTEGER;

            BEGIN
(1)             IF GoodsTotal > 20
(1)             THEN
(2)                 IF GoodsTotal > 100
(2)                 THEN
(3)                     IF HasDiscountCard = TRUE
(3)                     THEN
(3)                         Discount := 10
(3)                     ELSE
(3)                         Discount := 5
(2)                 ELSE
(4)                     IF HasDiscountCard = TRUE
(4)                     THEN
(4)                         Discount := 5
(4)                     ELSE
(4)                         Discount := 0
(1)                 ELSE
(1)                     Discount := 0;
            END;
```

Example Python

```
def Discount(GoodsTotal, HasDiscountCard) :

(1)   if GoodsTotal > 20:
(2)       if GoodsTotal > 100:
(3)           if HasDiscountCard == True:
(3)               return 10
(3)           else:
(3)               return 5
(2)       else:
(4)           if HasDiscountCard == TRUE:
(4)               return 5
(4)           else:
(4)               return 0
(1)   else:
(1)       return 0
```

[6]

Example candidate response – high

(a) Complete the decision table.

Conditions	goods totalling more than $20	Y	Y	Y	Y	N	N	N	N
	goods totalling more than $100	Y	Y	N	N	Y	Y	N	N
	have discount card	Y	N	Y	N	Y	N	Y	N
Actions	No discount				✓	✓	✓	✓	✓
	5% discount		✓	✓					
	10% discount	✓			[illegible]				

[4]

(b) Simplify your solution by removing redundancies.

Conditions	goods totalling more than $20	Y	Y	Y	Y	N			
	goods totalling more than $100	Y	Y	N	N	-			
	have discount card	Y	N	Y	N	-			
Actions	No discount				✓	✓			
	5% discount		✓	✓					
	10% discount	✓							

[5]

Example candidate response – high, continued

(c) The simplified table produced in **part (b)** is used as a design for program code.

The following identifier table shows the parameters to be passed to the function `Discount`. This function returns the discount amount as an integer.

Identifier	Data type
GoodsTotal	INTEGER
HasDiscountCard	BOOLEAN

Write **program code** for this function.

Programming language Visual BASIC

```
Function Discount(ByVal GoodsTotal As Integer, ByVal HasDiscountCard -
        As Boolean)
Dim Discount As Integer
If GoodsTotal < 20 Then
    Discount = 0
Else if GoodsTotal > 100 And HasDiscountCard = FALSE Then
    Discount = GoodsTotal x 5 ÷ 100
Else if GoodsTotal >20 And GoodsTotal <100 And HasDiscountCard=TRUE Then
    Discount = GoodsTotal x 5 ÷ 100
Elseif GoodsTotal >100 And HasDiscountCard = TRUE Then
    Discount = GoodsTotal x 10 ÷ 100
End if
  Return Discount
End Function
```

[6]

Examiner comment – high

In part (a) the candidate correctly ticked just one action for each combination of conditions.

In part (b) the candidate was able to distinguish which conditions could not be simplified. When the condition that goods totalled more than $20 was not satisfied then all other conditions were irrelevant and so could be replaced by a dash. So the candidate correctly replaced the four final columns in part (a) by just one column, removing redundancies.

In part (c) the candidate used Visual Basic to answer this part. The function header included the parameters but did not show the data type for the return value of the function. The candidate did not appreciate that a local variable of the same name as the function is not possible. The candidate tried to deal with the last column of the answer in part (b) first but did not realise that the opposite to testing for GoodsTotal > 20 is in fact testing for GoodsTotal <= 20. The candidate used nested IF statements to some extent, but could have improved on this rather than writing complex conditions involving AND.

Marks awarded for part (a) = 4/4
Marks awarded for part (b) = 5/5
Marks awarded for part (c) = 3/6

Total marks awarded = 12 out of 15

Example candidate response – middle

(a) Complete the decision table.

Conditions	goods totalling more than $20	Y	Y	Y	Y	N	N	N	N
	goods totalling more than $100	Y	Y	N	N	Y	Y	N	N
	have discount card	Y	N	Y	N	Y	N	Y	N
Actions	No discount				Y				Y
	5% discount		Y	Y			Y	Y	
	10% discount	Y				Y			

[4]

(b) Simplify your solution by removing redundancies.

Conditions	goods totalling more than $20	Y	Y	Y	N	N	N		
	goods totalling more than $100	Y	Y	N	Y	Y	N		
	have discount card	Y	N	N	Y	N	N		
Actions	No discount			Y			Y		
	5% discount		Y			Y			
	10% discount	Y			Y				

[5]

Example candidate response – middle, continued

(c) The simplified table produced in **part (b)** is used as a design for program code.

The following identifier table shows the parameters to be passed to the function `Discount`. This function returns the discount amount as an integer.

Identifier	Data type
GoodsTotal	INTEGER
HasDiscountCard	BOOLEAN

Write **program code** for this function.

Programming language ..... Python .....

```
def Distount (GoodsTotal , HasDiscountCard):
    if (GoodsTotal > 100 $) and (HasDiscountCard = "Y"):
        NewDiscount = int(GoodsTotal * 0.9 )
    elif (GoodsTotal < 100) and (HasDiscountCard = "Y"):
        NewDiscount = int (GoodsTotal * 0.95 )
    elif (GoodsTotal > 100) and (HasDiscountCard = "N"):
        NewDiscount = int(GoodsTotal * 0.95 )
    else :
        NewDiscount = GoodsTotal

Discount (x , y)
```

[6]

Examiner comment – middle

In part (a) the candidate identified the correct actions when the goods totalled more than $20, but did not appreciate that as soon as the goods do not total more than $20 no discount is given.

In part (b) the candidate recognised some of the conditions which could not be simplified, but did not notice that when goods do not total more than $20 then all other conditions are irrelevant as no discount is given.

In part (c) the candidate used Python to answer this part. The code is correctly indented and the nested IF statements correctly formed. This suggests practical programming experience. The function header correctly showed the parameters but the function body does not return a result. The question stated that the parameter HasDiscountCard is Boolean. However, the candidate handles the parameter values as though they were character values Y or N. The candidate does not check whether the goods total is over $20 but less than or equal to $100.

Marks awarded for part (a) = 3/4
Marks awarded for part (b) = 3/5
Marks awarded for part (c) = 2/6

Total marks awarded = 8 out of 15

Example candidate response – low

(a) Complete the decision table.

Conditions	goods totalling more than $20	Y	Y	Y	Y	N	N	N	N
	goods totalling more than $100	Y	Y	N	N	Y	Y	N	N
	have discount card	Y	N	Y	N	Y	N	Y	N
Actions	No discount				Y				Y
	5% discount		Y	Y			Y	Y	
	10% discount	Y				Y			

[4]

(b) Simplify your solution by removing redundancies.

Conditions	goods totalling more than $20	–	–	–	–				
	goods totalling more than $100	Y	Y	N	N				
	have discount card	Y	N	Y	N				
Actions	No discount				Y				
	5% discount		Y	Y					
	10% discount	Y							

[5]

Example candidate response – low, continued

(c) The simplified table produced in **part (b)** is used as a design for program code.

The following identifier table shows the parameters to be passed to the function `Discount`. This function returns the discount amount as an integer.

Identifier	Data type
GoodsTotal	INTEGER
HasDiscountCard	BOOLEAN

Write **program code** for this function.

Programming language ..... Python .....

```
def Discount ( GoodsTotal, HasDiscountCard) :
    if GoodsTotal () > 100 and HasDiscountCard() == "Y" :
        DiscountAmount = 10
    elif GoodsTotal()<100 and HasDiscountCard() == "Y" :
        DiscountAmount = 5
    elif GoodsTotal()>100 and HasDiscountCard() == "N" :
        DiscountAmount = 5
    else :
        DiscountAmount = 0.

    return DiscountAmount
```

..... [6]

Examiner comment – low

In part (a) the candidate identified the correct actions when the goods totalled more than $20, but did not appreciate that as soon as the goods do not total more than $20 no discount is given.

In part (b) the response is a deduction from part (a). However, the candidate needed to check whether this answer makes sense. The question stated that discount is only given when goods total more than $20. So completing the first row with the 'don't care' symbol is clearly incorrect.

In part (c) the candidate used Python to answer this part. The function header correctly showed the parameters and the function body returns the calculated result. The question stated that the parameter HasDiscountCard is Boolean. However, the candidate handles the parameter values as though they were character values Y or N. The candidate does not check whether the goods total is over $20 but less than or equal to $100.

Marks awarded for part (a) = 3/4
Marks awarded for part (b) = 0/5
Marks awarded for part (c) = 2/6

Total marks awarded = 5 out of 15

Question 4

4 A payroll program is to be written using an object-oriented programming language. An `Employee` class is designed. Two subclasses have been identified:

- `HourlyPaidEmployee` who is paid a monthly wage calculated from their hourly rate of pay and the number of hours worked during the month
- `SalariedEmployee` who is paid a monthly wage which is one 12th of their annual salary

(a) Draw an inheritance diagram for these classes.

[3]

(b) The design for the `Employee` class consists of:

- properties
  - `EmployeeName`
  - `EmployeeID`
  - `AmountPaidThisMonth`
- methods
  - `SetEmployeeName`
  - `SetEmployeeID`
  - `CalculatePay`

Write **program code** for the class definition of the superclass `Employee`.

Programming language .....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [5]

Question 4, continued

(c) (i) State the properties and/or methods required for the subclass `HourlyPaidEmployee`.

.....

.....

.....

..... [4]

(ii) State the properties and/or methods required for the subclass `SalariedEmployee`.

.....

.....

.....

..... [2]

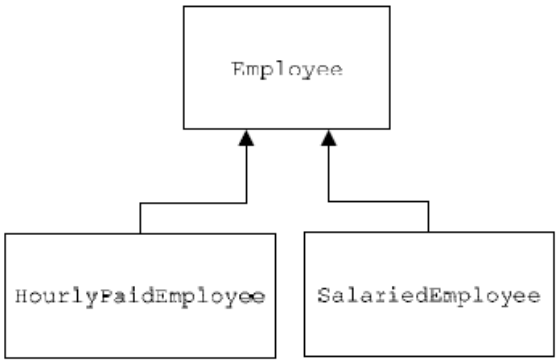
(d) Name the feature of object-oriented program design that allows the method `CalculatePay` to be declared in the superclass `Employee`.

.....

..... [1]

Mark scheme

4 (a)



[3]

(b) Example Pascal

```
Type
Employee = CLASS
        PUBLIC
            procedure SetEmployeeName
            Procedure SetEmployeeID
            Procedure CalculatePay
        PRIVATE
            EmployeeName : STRING
            EmployeeID : STRING
            AmountPaidThisMonth : Currency
    END;
```

Mark as follows:

Class header	(1 mark)
`PUBLIC` and `PRIVATE` used correctly	(1 mark)
`EmployeeName` + `EmployeeID`	(1 mark)
`AmountPaidThisMonth`	(1 mark)
Methods x 3	(1 mark)

Example VB

```
Class Employee
    Private EmployeeName As String
    Private EmployeeID As String
    Private AmountPaidThisMonth As Decimal
Public Sub SetEmployeeName()
End Sub
Public Sub SetEmployeeID()
End Sub
Public Sub CalculatePay()
End Sub
```

Example Python

```
Class Employee():
    def __init__(self):
        self.__EmployeeName = ""
        self.__EmployeeID = ""
        self.__AmountPaidThisMonth = 0
    def SetEmployeeName(self, Name):
        self.__EmployeeName = Name
    def SetEmployeeID(self, ID):
        self.__EmployeeID = ID
    def SetAmountPaidThisMonth(self, Paid):
        self.__AmountPaidThisMonth = Paid
```

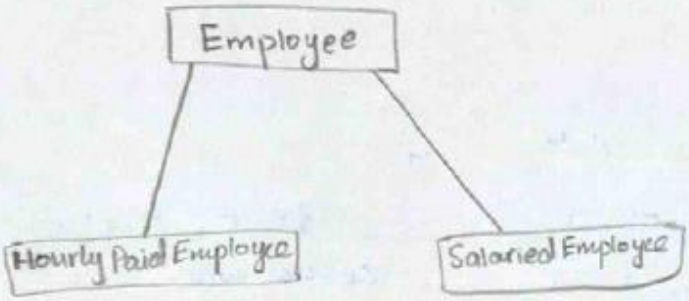
[max 5]

Mark scheme, continued

(c)	(i)	`HoursWorked`	1	
		`HourlyPayRate`	1	
		`SetHoursWorked`	1	
		`CalculatePay : Override`	1 + 1	
		`SetPayRate`	1	[max 4]
	(ii)	`AnnualSalary`	1	
		`SetSalary`	1	
		`CalculatePay : Override`	1	[max 2]
(d)	Polymorphism			[1]

Example candidate response – high

(a) Draw an inheritance diagram for these classes.



[3]

(b) The design for the `Employee` class consists of:

- properties
  - `EmployeeName`
  - `EmployeeID`
  - `AmountPaidThisMonth`
- methods
  - `SetEmployeeName`
  - `SetEmployeeID`
  - `CalculatePay`

Write **program code** for the class definition of the superclass `Employee`.

Programming language ..... e.g. .....

```
Class Employee
Private EmployeeName, EmployeeID as String
Private AmountPaidThisMonth as integer
Public sub setEmployeeName (Byval s as string)
    EmployeeName = S                                    END SUB
Public sub setEmployeeID( Byval i as string)
    EmployeeID = i                                      END SUB
Public function calculatePay() as integer
END functi
END class
```

[5]

Example candidate response – high, continued

(c) (i) State the properties and/or methods required for the subclass `HourlyPaidEmployee`.

The subclass HourlyPaidEmployee will inherit all properties and methods from the superclass Employee. The subclass will have hourly rate of pay and number of hours worked as properties. [4]

(ii) State the properties and/or methods required for the subclass `SalariedEmployee`.

The subclass SalariedEmployee will also inherit all properties and methods from the superclass employee. The properties required for this subclass will be the annual salary of the employee. [2]

(d) Name the feature of object-oriented program design that allows the method `CalculatePay` to be declared in the superclass `Employee`.

Polymorphism

[1]

Examiner comment – high

In part (a) the candidate correctly drew the superclass and the two subclasses, and connected each subclass to the superclass. The candidate did not complete the inheritance diagram showing the relationship with an arrow pointing from the subclass to the superclass.

In part (b) the candidate should have stated the programming language as VB.Net in the first line of the answer space. The candidate shows excellent knowledge of how to declare a superclass.

In parts (c)(i) and (ii) the candidate correctly states the extra properties required for the subclasses HourlyPaidEmployee and SalariedEmployee but does not state the additional methods required.

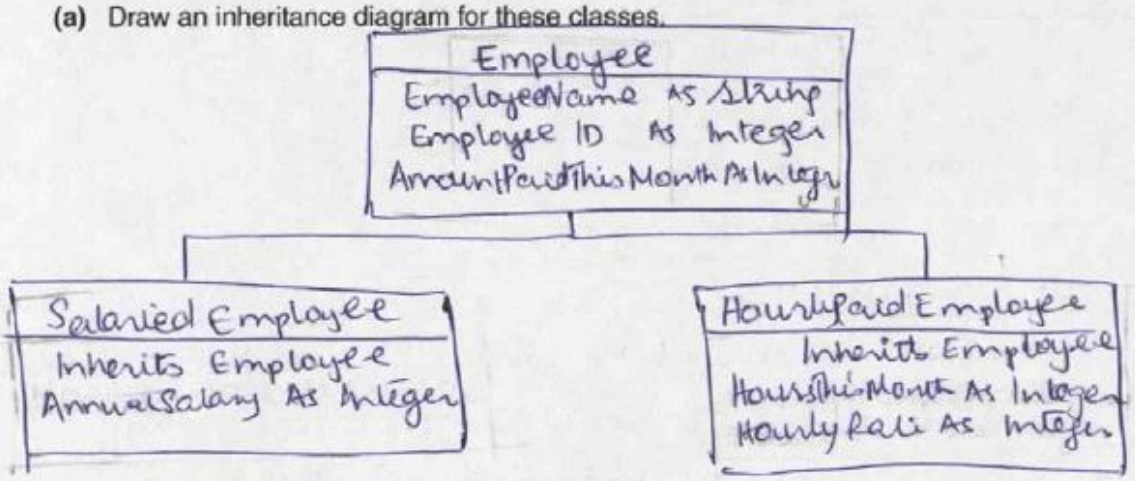
In part (d) the candidate recognised that the feature used here is polymorphism.

Marks awarded for part (a) = 2/3
Marks awarded for part (b) = 5/5
Marks awarded for part (c) = (i) 2/4, (ii) 1/2
Marks awarded for part (d) = 1/1

Total marks awarded = 11 out of 15

Example candidate response – middle

(a) Draw an inheritance diagram for these classes.



[3]

(b) The design for the `Employee` class consists of:

- properties
  - `EmployeeName`
  - `EmployeeID`
  - `AmountPaidThisMonth`
- methods
  - `SetEmployeeName`
  - `SetEmployeeID`
  - `CalculatePay`

Write **program code** for the class definition of the superclass `Employee`.

Programming language ....VB....

```
Public Class Employee
Dim EmployeeName As String
Dim Employee ID As Integer
Dim AmountPaidThisMonth As Integer
Overridable Sub Calculate Pay()
End Sub
EmployeeName = Console.ReadLine
Employee ID = Console.ReadLine
EndClass
```

[5]

Example candidate response – middle, continued

(c) (i) State the properties and/or methods required for the subclass `HourlyPaidEmployee`.

inherits →

```
Dim HoursThisMonth, HourlyRate As Integer
Override Sub CalculatePay
AmountPaidThisMonth = HoursThisMonth * HourlyRate
End Sub
```

[4]

(ii) State the properties and/or methods required for the subclass `SalariedEmployee`.

Inherits →

```
Dim AnnualSalary As Integer
Override Sub CalculatePay
AmountPaidThisMonth = 1/12 * AnnualSalary
End Sub
```

[2]

(d) Name the feature of object-oriented program design that allows the method `CalculatePay` to be declared in the superclass `Employee`.

Overriding / refernal

[1]

Examiner comment – middle

In part (a) the candidate correctly drew the superclass and the two subclasses, and connected each subclass to the superclass. The candidate did not complete the inheritance diagram showing the relationship with an arrow pointing from the subclass to the superclass. The inclusion of the properties is not required for an inheritance diagram.

In part (b) the candidate displayed some knowledge of how to declare a class in VB.Net. The class heading and ending is correct and one method is provided, although three methods were required as part of the question. The properties of a class should be declared using the keyword Private rather than Dim.

In parts (c)(i) and (ii) the candidate correctly states the extra properties required for the subclasses HourlyPaidEmployee and SalariedEmployee. The candidate also realised that the method CalculatePay needs to be redeclared for each of the subclasses.

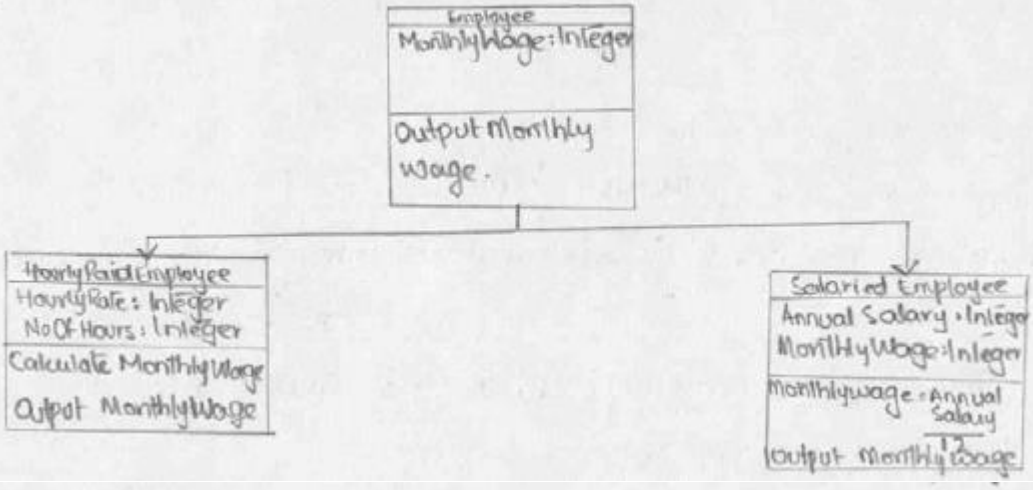
In part (d) by stating 'overriding' the candidate demonstrates some understanding of what the question is asking but this is not the correct term required here.

Marks awarded for part (a) = 2/3
Marks awarded for part (b) = 2/5
Marks awarded for part (c) = (i) 3/4, (ii) 2/2
Marks awarded for part (d) = 0/1

Total marks awarded = 9 out of 15

Example candidate response – low

(a) Draw an inheritance diagram for these classes.



[3]

(b) The design for the `Employee` class consists of:

- properties
  - `EmployeeName`
  - `EmployeeID`
  - `AmountPaidThisMonth`
- methods
  - `SetEmployeeName`
  - `SetEmployeeID`
  - `CalculatePay`

Write **program code** for the class definition of the superclass `Employee`.

Programming language ..... VB.Net 2010.

```
Structure Employee
    Dim EmployeeName as String
    Dim Employee ID as Integer
    Dim Amount Paid This Month as Integer
End Structure
console.writeline ("Enter Employee Name")
Employee. Name = console. readline
console. writeline("Enter Employee ID")
Employee. Employee ID = console. readline
console. writeline ( Amount Paid This Month )
```

[5]

Example candidate response – low, continued

(c) (i) State the properties and/or methods required for the subclass `HourlyPaidEmployee`.

The properties for HourlyPaidEmployee will be the EmployeeName, EmployeeID, HourlyRate of Pay and NumberOf Hours worked. Whereas the method will be calculated using the HourlyRate and NumberOfHours worked. [4]

(ii) State the properties and/or methods required for the subclass `SalariedEmployee`.

The properties will be the EmployeeName, EmployeeID and AnnualSalary. The monthlywage will be calculated by dividing the AnnualSalary by 12. [2]

(d) Name the feature of object-oriented program design that allows the method `CalculatePay` to be declared in the superclass `Employee`.

Encapsulation

[1]

Examiner comment – low

In part (a) the candidate correctly drew the superclass and the two subclasses, and connected each subclass to the superclass. The candidate did not complete the inheritance diagram showing the relationship with an arrow pointing from the subclass to the superclass. The inclusion of properties and methods is not required for an inheritance diagram.

In part (b) the candidate does not appear to have any knowledge of how to declare a class in VB.Net. The answer given here is an attempt at the declaration of a record. The keyword Dim is not appropriate here.

In parts (c)(i) and (ii) the candidate included the properties required for the subclasses HourlyPaidEmployee and SalariedEmployee as well as the inherited properties which did not need listing here. The candidate misinterpreted the term 'method' and described how the calculation of pay should be performed rather than giving the identifier of the subclass methods required.

In part (d) the candidate provides an object-oriented programming term that applies to all classes, not the required term for methods that behave differently for different subclasses.

Marks awarded for part (a) = 2/3
Marks awarded for part (b) = 1/5
Marks awarded for part (c) = (i) 2/4, (ii) 1/2
Marks awarded for part (d) = 0/1

Total marks awarded = 6 out of 15

Question 5

5 Data is stored in the array `NameList[1:10]`. This data is to be sorted.

(a) (i) Complete the pseudocode algorithm for an insertion sort.

```
FOR ThisPointer ← 2 TO ……
    // use a temporary variable to store item which is to
    // be inserted into its correct location
    Temp ← NameList[ThisPointer]
    Pointer ← ThisPointer - 1

    WHILE (NameList[Pointer] > Temp) AND ……
        // move list item to next location
        NameList[……] ← NameList[……]
        Pointer ← ……
    ENDWHILE

    // insert value of Temp in correct location
    NameList[……] ← ……
ENDFOR
```

[7]

(ii) A special case is when `NameList` is already in order. The algorithm in **part (a)(i)** is applied to this special case.

Explain how many iterations are carried out for each of the loops.

……

……

……

……

……

…… [3]

Question 5, continued

(b) An alternative sort algorithm is a bubble sort:

```
FOR ThisPointer ← 1 TO 9
   FOR Pointer ← 1 TO 9
       IF NameList[Pointer] > NameList[Pointer + 1]
          THEN
             Temp ← NameList[Pointer]
             NameList[Pointer] ← NameList[Pointer + 1]
             NameList[Pointer + 1] ← Temp
       ENDIF
    ENDFOR
ENDFOR
```

(i) As in **part (a)(ii)**, a special case is when `NameList` is already in order. The algorithm in **part (b)** is applied to this special case.

Explain how many iterations are carried out for each of the loops.

.....

.....

.....

..... [2]

Question 5, continued

(ii) Rewrite the algorithm in **part (b)**, using **pseudocode**, to reduce the number of unnecessary comparisons. Use the same variable names where appropriate.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [5]

Mark scheme

5 (a) (i)

```
FOR ThisPointer ← 2 TO 10
    // use a temporary variable to store item which is to
    // be inserted into its correct location
    Temp ← NameList[ThisPointer]
    Pointer ← ThisPointer - 1

    WHILE (NameList[Pointer] > Temp) AND (Pointer > 0)
        // move list item to next location
        NameList[Pointer + 1] ← NameList[Pointer]
        Pointer ← Pointer - 1
    ENDWHILE

    // insert value of Temp in correct location
    NameList[Pointer + 1] Temp←
ENDFOR
```

1 mark for each gap filled correctly **[7]**

(ii)

The outer loop (FOR loop) is executed 9 times	(1 mark)	
it is not dependant on the dataset	(1 mark)	
The Inner loop (WHILE loop) is not entered	(1 mark)	
as the condition is already false at the first encounter	(1 mark)	**[max 3]**

(b) (i)

outer loop is executed 9 times	(1 mark)	
inner loop is executed 9 times (for each iteration of the outer loop)	(1 mark)	
not dependant on the dataset	(1 mark)	**[max 2]**

(ii)

```
NumberOfItems ← 10
REPEAT
    NoMoreSwaps    ← TRUE

    FOR Pointer ← 1 TO NumberOfItems - 1
        IF NameList[Pointer] > NameList[Pointer + 1]
            THEN
                NoMoreSwaps ← FALSE
                Temp ← NameList[Pointer]
                NameList[Pointer] ← NameList[Pointer + 1]
                NameList[Pointer + 1] ← Temp
            ENDIF
        ENDFOR
        NumberOfItems ← NumberOfItems - 1
UNTIL NoMoreSwaps = TRUE
```

Mark as follows:

• change outer loop to a REPEAT/WHILE loop	(1 mark)	
• FOR loop has variable used for final value	(1 mark)	
• Initialise Boolean variable to TRUE	(1 mark)	
• set Boolean variable to FALSE in correct place	(1 mark)	
• number of items to consider on each pass decrements	(1 mark)	
• Correct stopping condition for REPEAT loop	(1 mark)	**[max 5]**

Example candidate response – high

5 Data is stored in the array `NameList[1:10]`. This data is to be sorted.

(a) (i) Complete the pseudocode algorithm for an insertion sort.

```
FOR ThisPointer ← 2 TO 10
    // use a temporary variable to store item which is to
    // be inserted into its correct location
    Temp ← NameList[ThisPointer]
    Pointer ← ThisPointer - 1

    WHILE (NameList[Pointer] > Temp) AND Pointer > 0 1
        // move list item to next location
        NameList[Pointer + 1] ← NameList[Pointer]
        Pointer ← Pointer - 1
    ENDWHILE

    // insert value of Temp in correct location
    NameList[Pointer + 1] ← Temp
ENDFOR
```

[7]

(ii) A special case is when `NameList` is already in order. The algorithm in part (a)(i) is applied to this special case.

Explain how many iterations are carried out for each of the loops.

One because the WHILE loop is not executed as the NameList[Pointer] will always be be less than Temp

[3]

Example candidate response – high, continued

(b) An alternative sort algorithm is a bubble sort:

```
FOR ThisPointer ← 1 TO 9
    FOR Pointer ← 1 TO 9
        IF NameList[Pointer] > NameList[Pointer + 1]
            THEN
                Temp ← NameList[Pointer]
                NameList[Pointer] ← NameList[Pointer + 1]
                NameList[Pointer + 1] ← Temp
        ENDIF
    ENDFOR
ENDFOR
```

(i) As in **part (a)(ii)**, a special case is when `NameList` is already in order. The algorithm in **part (b)** is applied to this special case.

Explain how many iterations are carried out for each of the loops.

Nine because they are nested for loop FOR loops, so execute whatever the conditions of the list are

[2]

Example candidate response – high, continued

(ii) Rewrite the algorithm in **part (b)**, using **pseudocode**, to reduce the number of unnecessary comparisons. Use the same variable names where appropriate.

```
Sorted = FALSE
Pointer Length = 9
                                    Pointer
WHILE NOT Sorted AND Length > 1
Sorted = TRUE
FOR Pointer Index ← 1 TO Pointer
    IF NameList[Index] > NameList[Index+1]
    THEN
        Temp ← NameList[Index]
        NameList[Index] ← NameList[Index+1]
        NameList[Index+1] ← Temp
        Sorted = False
    ENDIF
                Pointer ENDFOR
Pointer POINTER ← Pointer - 1
ENDWHILE
```

[5]

Examiner comment – high

In parts (a)(i) and (ii) the candidate demonstrates excellent understanding of the insertion sort algorithm. The only inaccuracy is that the second condition for the WHILE loop would terminate the loop too early. The candidate correctly recognises that the WHILE loop will not execute at all when the NameList is already in order. However, the fact that the FOR loop will execute 9 times regardless of the state of NameList is not mentioned.

In part (b)(i) the response here is not very clear. The candidate needs to state that each FOR loop will execute 9 times as they are not dependent on any condition. In part (b)(ii) the candidate demonstrates excellent understanding of how a bubble sort operates and where changes are possible to improve efficiency. The candidate realises that the outer loop can be changed to a conditional loop, so it terminates when there are no more changes, and the inner loop does not need to examine every element every time. With each iteration of the outer loop, another element will be in its correct position and therefore the upper value of the FOR loop can be decreased.

Marks awarded for part (a) = (i) 6/7, (ii) 2/3
Marks awarded for part (b) = (i) 1/2, (ii) 5/5

Total marks awarded = 14 out of 17

Example candidate response – middle

5 Data is stored in the array `NameList[1:10]`. This data is to be sorted.

(a) (i) Complete the pseudocode algorithm for an insertion sort.

```
FOR ThisPointer ← 2 TO 10
    // use a temporary variable to store item which is to
    // be inserted into its correct location
    Temp ← NameList[ThisPointer]
    Pointer ← ThisPointer - 1
    WHILE (NameList[Pointer] > Temp) AND (Pointer >= 1)
        // move list item to next location
        NameList[ThisPointer] ← NameList[Pointer]
        Pointer ← Pointer - 1
    ENDWHILE
    // insert value of Temp in correct location
    NameList[Pointer] ← Temp
ENDFOR
```

[7]

(ii) A special case is when `NameList` is already in order. The algorithm in part (a)(i) is applied to this special case.

Explain how many iterations are carried out for each of the loops.

No ~~9 nine~~ iteration of while loop A will be carried out ~~Although~~ the data may be is in order ~~but~~ the program ~~has~~ still condition of 'Name List(Pointer) > temp' will not be fulfilled. Only For loop will be carried out nine times. [3]

Example candidate response – middle, continued

(b) An alternative sort algorithm is a bubble sort:

```
FOR ThisPointer ← 1 TO 9
    FOR Pointer ← 1 TO 9
        IF NameList[Pointer] > NameList[Pointer + 1]
            THEN
                Temp ← NameList[Pointer]
                NameList[Pointer] ← NameList[Pointer + 1]
                NameList[Pointer + 1] ← Temp
        ENDIF
    ENDFOR
ENDFOR
```

(i) As in **part (a)(ii)**, a special case is when `NameList` is already in order. The algorithm in **part (b)** is applied to this special case.

Explain how many iterations are carried out for each of the loops.

64 iteration loops will be carried. The program is going to compare each data item with the rest of the data items eventhough they are already in order. [2]

Example candidate response – middle, continued

(ii) Rewrite the algorithm in **part (b)**, using **pseudocode**, to reduce the number of unnecessary comparisons. Use the same variable names where appropriate.

```
For this pointer = 1 to 9
For Pointer = 1 to 9 - This Pointer
If [NameList(Pointer)] > [NameList(Pointer + 1)]
then
    Temp = NameList(Pointer)
NameList(Pointer) = NameList(Pointer + 1)
NameList(Pointer + 1) = Temp
End if
End For
End For
```

[5]

Examiner comment – middle

In part (a)(i) the candidate demonstrates very good understanding of the insertion sort algorithm. There is some confusion over which pointer to use when moving a list item to the correct location. In part (a)(ii) the candidate correctly recognises that the WHILE loop will not execute at all when the NameList is already in order and that the FOR loop will execute nine times regardless of the state of NameList.

In parts (b)(i) and (ii) the candidate appears to have some understanding that for each iteration of the outer loop the inner loop is executed a set number of times. The misunderstanding that each loop iterates 8 times rather than 9 times results in the answer of 64 rather than a total of 81 iterations. The candidate realised that one way of making efficiency gains is to restrict the number of times the inner loop iterates for each iteration of the outer loop.

Marks awarded for part (a) = (i) 5/7, (ii) 3/3
Marks awarded for part (b) = (i) 0/2, (ii) 2/5

Total marks awarded = 10 out of 17

Example candidate response – low

5 Data is stored in the array `NameList[1:10]`. This data is to be sorted.

(a) (i) Complete the pseudocode algorithm for an insertion sort.

```
FOR ThisPointer ← 2 TO 10
    // use a temporary variable to store item which is to
    // be inserted into its correct location
    Temp ← NameList[ThisPointer]
    Pointer ← ThisPointer - 1
    WHILE (NameList[Pointer] > Temp) AND (Pointer <> 0)
        // move list item to next location
        NameList[Pointer] ← NameList[Temp]
        Pointer ← Pointer - 1
    ENDWHILE

    // insert value of Temp in correct location
    NameList[ThisPointer] ← Temp
ENDFOR
```

[7]

(ii) A special case is when `NameList` is already in order. The algorithm in **part (a)(i)** is applied to this special case.

Explain how many iterations are carried out for each of the loops.

The WHILE loop never gets a chance to iterate as all the items in the list are in order.
The FOR loop however, runs 9 times as the pointer starts at 2 and increments till it reaches 10.

[3]

Example candidate response – low, continued

(b) An alternative sort algorithm is a bubble sort:

```
FOR ThisPointer ← 1 TO 9
   FOR Pointer ← 1 TO 9
      IF NameList[Pointer] > NameList[Pointer + 1]
         THEN
            Temp ← NameList[Pointer]
            NameList[Pointer] ← NameList[Pointer + 1]
            NameList[Pointer + 1] ← Temp
      ENDIF
   ENDFOR
ENDFOR
```

(i) As in **part (a)(ii)**, a special case is when `NameList` is already in order. The algorithm in **part (b)** is applied to this special case.

Explain how many iterations are carried out for each of the loops.

The second FOR loop iterates 9 times, but the first FOR loop iterates only once. The second FOR loop checks that a name is not greater than the following name and since the list is in order, it simply moves on to compare the next 2 positions. [2]

Example candidate response – low, continued

(ii) Rewrite the algorithm in **part (b)**, using **pseudocode**, to reduce the number of unnecessary comparisons. Use the same variable names where appropriate.

```
FOR ThisPointer ← 1 TO 9
    FOR Pointer ← 1 TO 8
        IF NameList[Pointer] > NameList[Pointer + 1]
        THEN
            Temp ← NameList[Pointer]
            NameList[Pointer] ← NameList[Pointer + 1]
            NameList[Pointer + 1] ← Temp
        ENDIF
    ENDFOR
ENDFOR
```

[5]

Examiner comment – low

In part (a)(i) the candidate clearly understands that the FOR loop needs to iterate once for every list item except the first one. The complex condition of the WHILE loop is correct although the candidate is not clear how to move a list item to the next location. The candidate seems unaware that although Pointer is set to one less than ThisPointer before the WHILE loop, the value of Pointer changes within the WHILE loop. Therefore, when moving the contents of Temp to the correct location, (Pointer +1) must be used as index, not ThisPointer. In part (a)(ii) the candidate understands that the WHILE loop is not entered when NameList is in order, although the explanation is a little too vague. The candidate recognises that the FOR loop will always iterate 9 times.

In parts (b)(i) and (ii) the candidate states that the inner FOR loop iterates 9 times. This is true, but it will do so for each of the 9 times that the outer FOR loop iterates. The candidate seems to be unaware of this. Consequently the revised pseudocode is not an improved solution for the bubblesort algorithm. The only difference to the original pseudocode is that the inner loop does not address the 9th element. This means that the last element will not necessarily be in the correct position.

Marks awarded for part (a) = (i) 4/7, (ii) 2/3
Marks awarded for part (b) = (i) 1/2, (ii) 0/5

Total marks awarded = 7 out of 17

Question 6

6 A queue Abstract Data Type (ADT) has these associated operations:

- create queue
- add item to queue
- remove item from queue

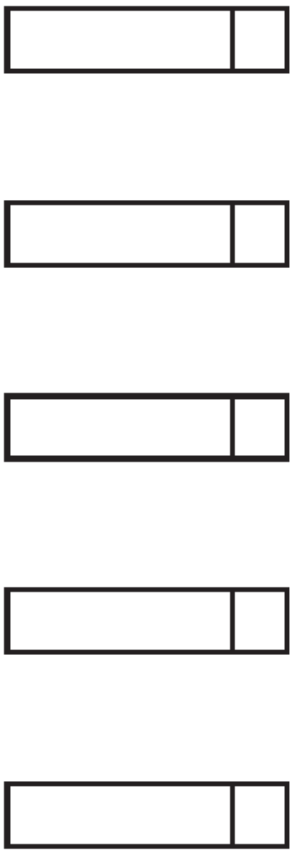
The queue ADT is to be implemented as a linked list of nodes.

Each node consists of data and a pointer to the next node.

(a) The following operations are carried out:

```
CreateQueue
AddName("Ali")
AddName("Jack")
AddName("Ben")
AddName("Ahmed")
RemoveName
AddName("Jatinder")
RemoveName
```

Add appropriate labels to the diagram to show the final state of the queue. Use the space on the left as a workspace. Show your final answer in the node shapes on the right:



[3]

Question 6, continued

(b) Using pseudocode, a record type, `Node`, is declared as follows:

```
TYPE Node
    DECLARE Name    : STRING
    DECLARE Pointer : INTEGER
ENDTYPE
```

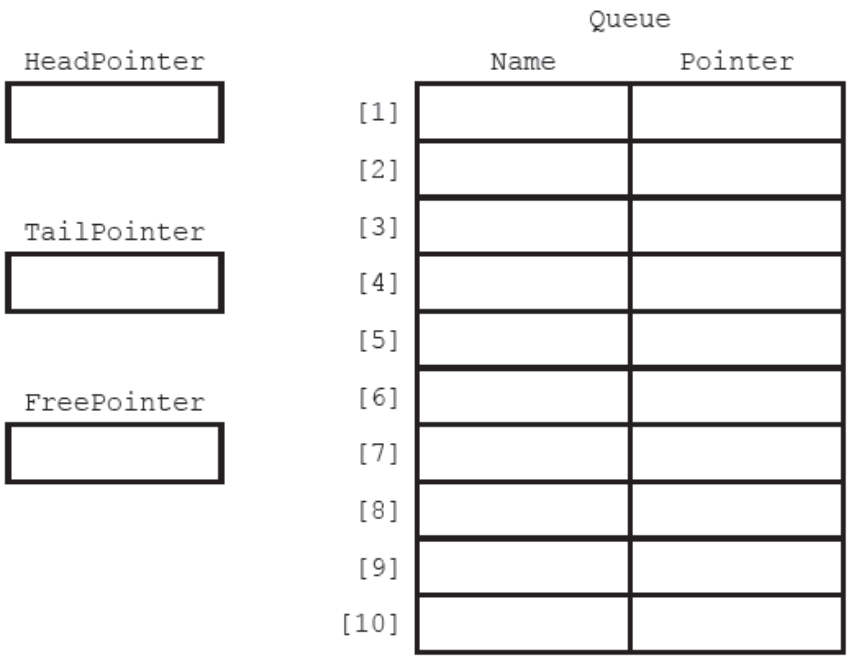
The statement

```
DECLARE Queue : ARRAY[1:10] OF Node
```

reserves space for 10 nodes in array `Queue`.

(i) The `CreateQueue` operation links all nodes and initialises the three pointers that need to be used: `HeadPointer`, `TailPointer` and `FreePointer`.

Complete the diagram to show the value of all pointers after `CreateQueue` has been executed.



[4]

Question 6, continued

(ii) The algorithm for adding a name to the queue is written, using pseudocode, as a procedure with the header:

```
PROCEDURE AddName(NewName)
```

where `NewName` is the new name to be added to the queue.

The procedure uses the variables as shown in the identifier table.

Identifier	Data type	Description
`Queue`	`Array[1:10] OF Node`	Array to store node data
`NewName`	`STRING`	Name to be added
`FreePointer`	`INTEGER`	Pointer to next free node in array
`HeadPointer`	`INTEGER`	Pointer to first node in queue
`TailPointer`	`INTEGER`	Pointer to last node in queue
`CurrentPointer`	`INTEGER`	Pointer to current node

```
PROCEDURE AddName(BYVALUE NewName : STRING)
   // Report error if no free nodes remaining
   IF FreePointer = 0
      THEN
         Report Error
   ELSE
      // new name placed in node at head of free list
      CurrentPointer ← FreePointer
      Queue[CurrentPointer].Name ← NewName
      // adjust free pointer
      FreePointer ← Queue[CurrentPointer].Pointer
      // if first name in queue then adjust head pointer
      IF HeadPointer = 0
         THEN
            HeadPointer ← CurrentPointer
      ENDIF
      // current node is new end of queue
      Queue[CurrentPointer].Pointer ← 0
      TailPointer ← CurrentPointer
   ENDIF
ENDPROCEDURE
```

Question 6, continued

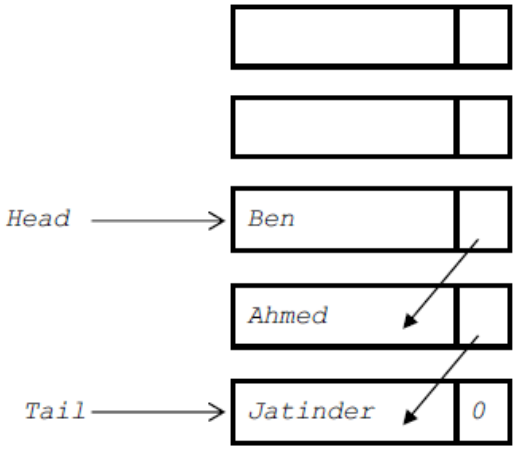
Complete the **pseudocode** for the procedure `RemoveName`. Use the variables listed in the identifier table.

```
PROCEDURE RemoveName()
   // Report error if Queue is empty
.....
.....
.....
.....
   OUTPUT Queue[.....].Name
   // current node is head of queue
.....
   // update head pointer
.....
   // if only one element in queue then update tail pointer
.....
.....
.....
.....
   // link released node to free list
.....
.....
.....
ENDPROCEDURE
```

[6]

Mark scheme

6 (a)



1 mark for Head and Tail pointers
1 mark for 3 correct items – linked as shown
1 mark for correct order with null pointer in last nod **[3]**

(b) (i)

Queue

	HeadPointer		Name	Pointer
	0	[1]		2
		[2]		3
	TailPointer	[3]		4
	0	[4]		5
		[5]		6
	FreePointer	[6]		7
	1	[7]		8
		[8]		9
		[9]		10
		[10]		0

Mark as follows:

HeadPointer =0 & TailPointer = 0
FreePointer assigned a value
Pointers[1] to [9] links the nodes together
Pointer[10] = 'Null' **[4]**

Mark scheme, continued

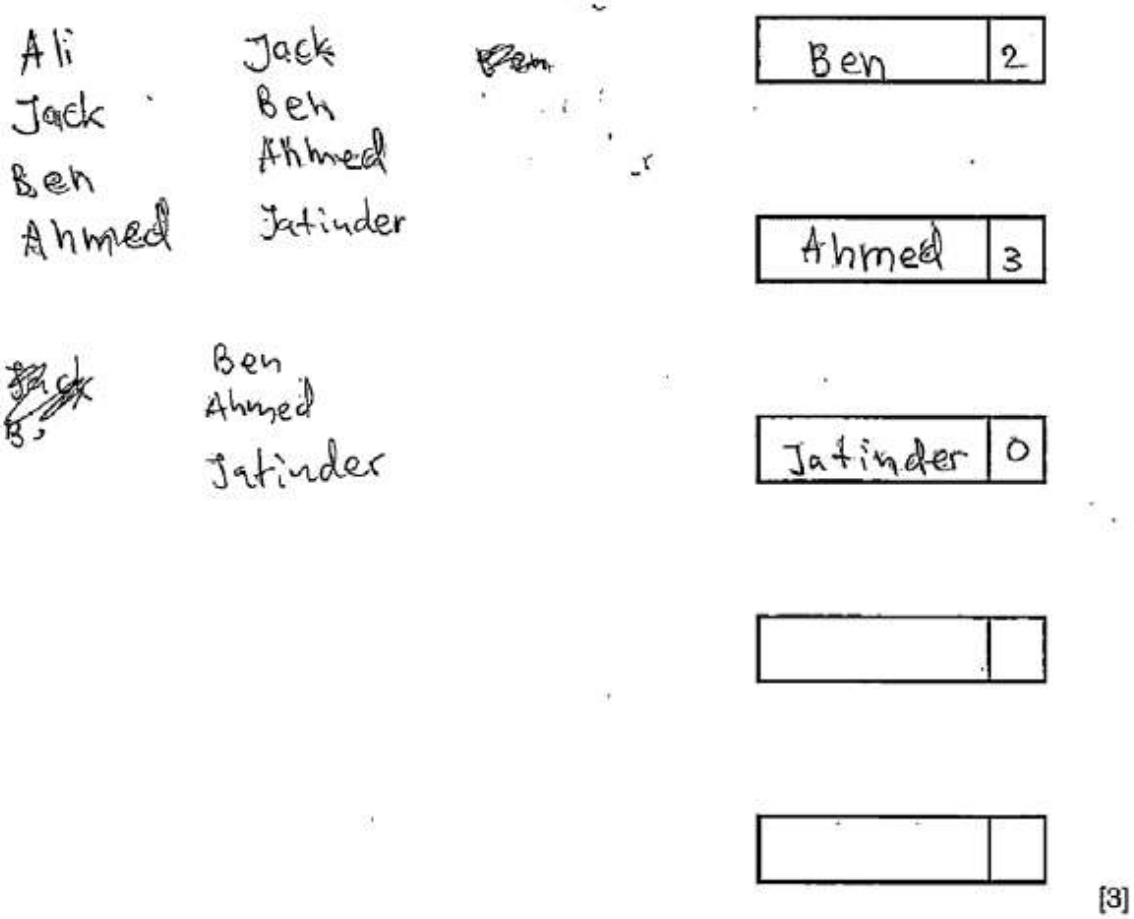
(ii)
```
PROCEDURE RemoveName()
    // Report error if Queue is empty
    IF HeadPointer = 0
        THEN
            Error
        ELSE
            OUTPUT Queue[HeadPointer].Name
            // current node is head of queue
            CurrentPointer ← HeadPointer
            // update head pointer
            HeadPointer ← Queue[CurrentPointer].Pointer
            //if only one element in queue,then update tail pointer
            IF HeadPointer = 0
                THEN
                    TailPointer ← 0
            ENDIF
                // link released node to free list
                Queue[CurrentPointer].Pointer ← FreePointer
            FreePointer ← CurrentPointer
    ENDIF
ENDPROCEDURE
```
[max 6]

Example candidate response – high

(a) The following operations are carried out:

```
CreateQueue
AddName("Ali")
AddName("Jack")
AddName("Ben")
AddName("Ahmed")
RemoveName
AddName("Jatinder")
RemoveName
```

Add appropriate labels to the diagram to show the final state of the queue. Use the space on the left as a workspace. Show your final answer in the node shapes on the right:



[3]

Example candidate response – high, continued

(b) Using pseudocode, a record type, `Node`, is declared as follows:

```
TYPE Node
    DECLARE Name    : STRING
    DECLARE Pointer : INTEGER
ENDTYPE
```

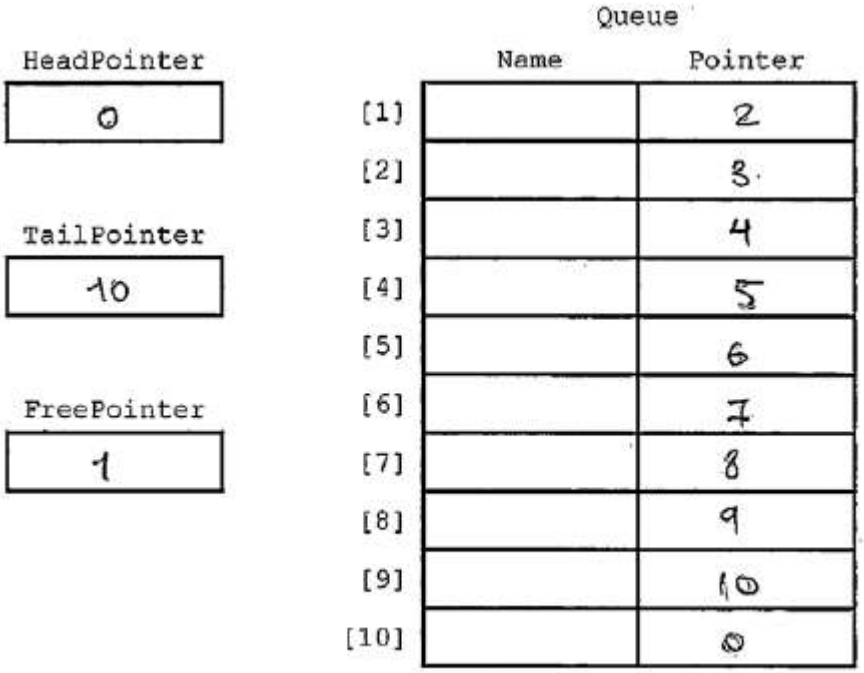
The statement

```
DECLARE Queue : ARRAY[1:10] OF Node
```

reserves space for 10 nodes in array `Queue`.

(i) The `CreateQueue` operation links all nodes and initialises the three pointers that need to be used: `HeadPointer`, `TailPointer` and `FreePointer`.

Complete the diagram to show the value of all pointers after `CreateQueue` has been executed.



[4]

Example candidate response – high, continued

Complete the **pseudocode** for the procedure `RemoveName`. Use the variables listed in the identifier table.

```
PROCEDURE RemoveName()
  // Report error if Queue is empty
IF HeadPointer = 0
    THEN
        Report Error
ELSE
  OUTPUT Queue[HeadPointer].Name
  // current node is head of queue
  CurrentPointer ← HeadPointer
  // update head pointer
HeadPointer ← Queue[CurrentPointer].Pointer
  // if only one element in queue then update tail pointer
IF TailPointer = 2
    THEN
        TailPointer ← CurrentPointer
ENDIF
  // link released node to free list
Queue[CurrentPointer].Pointer ← 0
FreePointer ← CurrentPointer
ENDPROCEDURE
```

[6]

Examiner comment – high

In part (a) the candidate used the work space to write the items in the queue at different points. The answer in the node shapes shows the correct names. However, the nodes have not been given labels, so it is open to interpretation that the pointer values point to the correct nodes. The candidate correctly shows the final node containing the name Jatinder with a null pointer. There are no pointers to show where the head and tail of the queue are.

In part (b)(i) when the CreateQueue operation has been carried out, the queue is empty and all nodes are part of the free list. The candidate correctly links all nodes with a null pointer in the final node. FreePointer correctly points to the first node in the free list and HeadPointer is the null pointer. However, as there is no queue content, the tail pointer should also be a null pointer.In part (b)(ii) the candidate understands that testing for an empty queue means testing for HeadPointer to be the null pointer. The candidate correctly initialises CurrentPointer and updates HeadPointer by following the pointer of the current node. The candidate does not appreciate that if the last name in the queue has just been removed the head pointer would now be null and therefore the tail pointer would also need to be set to null. Linking the released node to the free list means linking it to the front of the list that FreePointer is pointing to. The candidate correctly updates FreePointer, but the released node is not linked to the front of the free list.

Marks awarded for part (a) = 2/3
Marks awarded for part (b) = (i) 3/4, (ii) 5/6

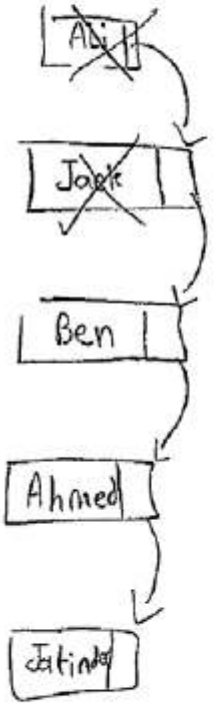
Total marks awarded = 10 out of 13

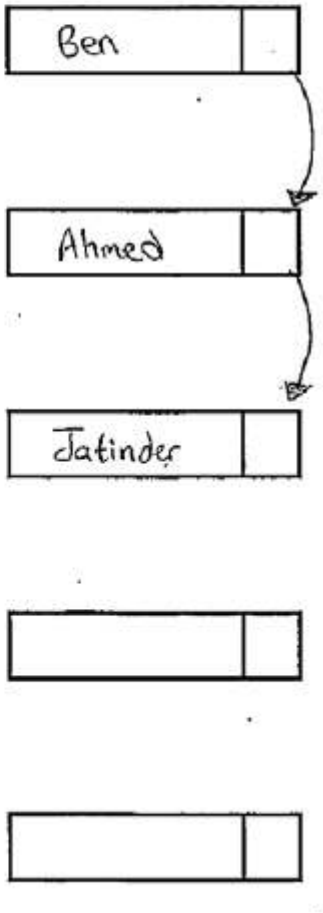
Example candidate response – middle

(a) The following operations are carried out:

```
CreateQueue
AddName("Ali")
AddName("Jack")
AddName("Ben")
AddName("Ahmed")
RemoveName
AddName("Jatinder")
RemoveName
```

Add appropriate labels to the diagram to show the final state of the queue. Use the space on the left as a workspace. Show your final answer in the node shapes on the right:





[3]

Example candidate response – middle, continued

(b) Using pseudocode, a record type, `Node`, is declared as follows:

```
TYPE Node
    DECLARE Name    : STRING
    DECLARE Pointer : INTEGER
ENDTYPE
```

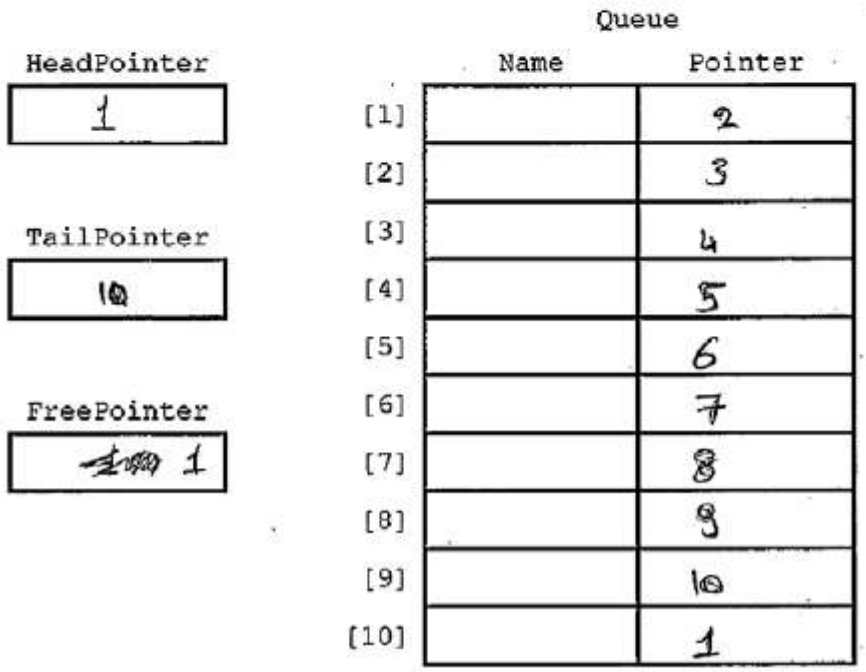
The statement

```
DECLARE Queue : ARRAY[1:10] OF Node
```

reserves space for 10 nodes in array `Queue`.

(i) The `CreateQueue` operation links all nodes and initialises the three pointers that need to be used: `HeadPointer`, `TailPointer` and `FreePointer`.

Complete the diagram to show the value of all pointers after `CreateQueue` has been executed.



[4]

Example candidate response – middle, continued

Complete the **pseudocode** for the procedure `RemoveName`. Use the variables listed in the identifier table.

```
PROCEDURE RemoveName()
  // Report error if Queue is empty
IF (HeadPointer = 0) THEN
        OUTPUT ERROR
ELSE
  OUTPUT Queue[.....HeadPointer.....].Name
  // current node is head of queue
        CurrentPointer ← HeadPointer
  // update head pointer
        HeadPointer = Queue[HeadPointer].Pointer
  // if only one element in queue then update tail pointer
IF HeadPointer = TailPointer THEN
        TailPointer ← CurrentPointer
End IF  TailPointer ← Queue[HeadPointer].Pointer
END IF
  // link released node to free list
Queue[CurrentPointer].Pointer = NULL
End IF
ENDPROCEDURE
```

[6]

Examiner comment – middle

(a) Enter your examiner comment here

In part (a) the candidate used the work space to draw the nodes in the queue at different points. The answer in the node shapes shows the correct names and the pointers point to the correct nodes. The final node containing the name Jatinder should have a null pointer. There are no pointers to show where the head and tail of the queue are.

In part (b)(i) when the CreateQueue operation has been carried out, the queue is empty and all nodes are part of the free list. The candidate correctly links all nodes. However, the final node does not contain a null pointer but points back to the beginning of the free list. FreePointer correctly points to the first node in the free list. However, HeadPointer and TailPointer should be null pointers as there is no queue content. In part (b)(ii) the candidate understands that testing for an empty queue means testing for HeadPointer to be the null pointer. The candidate correctly initialises CurrentPointer and updates HeadPointer by following the pointer of the current node. The candidate does not appreciate that if the last name in the queue has just been removed the head pointer would now be null and therefore the tail pointer would also need to be set to null. The candidate does not demonstrate that linking the released node to the free list means linking it to the front of the list that FreePointer is pointing to, so the pointer of the released node and FreePointer need to be updated.

Marks awarded for part (a) = 1/3
Marks awarded for part (b) = (i) 2/4, (ii) 4/6

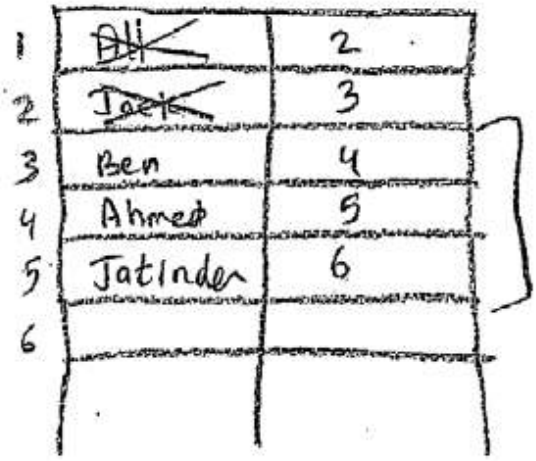
Total marks awarded = 7 out of 13

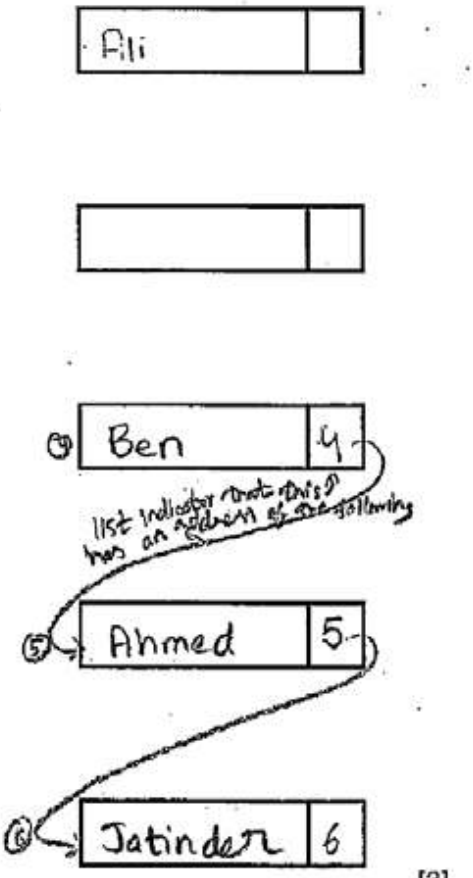
Example candidate response – low

(a) The following operations are carried out:

```
CreateQueue
AddName("Ali")
AddName("Jack")
AddName("Ben")
AddName("Ahmed")
RemoveName
AddName("Jatinder")
RemoveName
```

Add appropriate labels to the diagram to show the final state of the queue. Use the space on the left as a workspace. Show your final answer in the node shapes on the right:





[3]

Example candidate response – low, continued

(b) Using pseudocode, a record type, `Node`, is declared as follows:

```
TYPE Node
   DECLARE Name    : STRING
   DECLARE Pointer : INTEGER
ENDTYPE
```

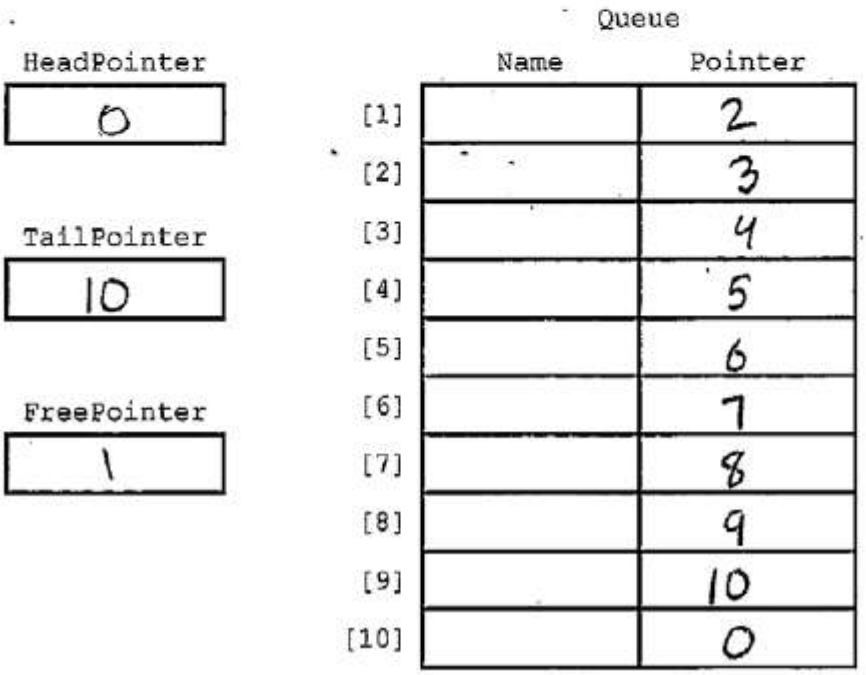
The statement

```
DECLARE Queue : ARRAY[1:10] OF Node
```

reserves space for 10 nodes in array `Queue`.

(i) The `CreateQueue` operation links all nodes and initialises the three pointers that need to be used: `HeadPointer`, `TailPointer` and `FreePointer`.

Complete the diagram to show the value of all pointers after `CreateQueue` has been executed.



[4]

Example candidate response – low, continued

Complete the **pseudocode** for the procedure RemoveName. Use the variables listed in the identifier table.

```
PROCEDURE RemoveName()
  // Report error if Queue is empty
  If FreePointer = 1
  THEN
      Report Error
  Else
  OUTPUT Queue[CurrentPointer].Name
  // current node is head of queue
  Queue(CurrentPointer).Pointer = 10
  // update head pointer
  HeadPointer = Queue(currentPointer).Pointer
  // if only one element in queue then update tail pointer
  If Queue[currentpointer].Pointer = 2 then
  TailPointer = 1
  End If

  // link released node to free list
  Queue Freelist ← Queue(currentpointer)

  End If
ENDPROCEDURE
```

[6]

Examiner comment – low

In part (a) the candidate used the work space to draw the nodes in the queue at different points. The answer in the node shapes shows the correct names and the pointers point to the correct nodes. The final node containing the name Jatinder should have a null pointer. There are no pointers to show where the head and tail of the queue are.

In part (b)(i) when the CreateQueue operation has been carried out, the queue is empty and all nodes are part of the free list. The candidate correctly links all nodes with a null pointer in the final node. FreePointer correctly points to the first node in the free list and HeadPointer is a null pointer. However, as there is no queue content, the tail pointer should also be a null pointer. The candidate seems to assume that if FreePointer points to the first node in the array, the queue must be empty. This is true when the queue is first initialised, but may not be true after names have been added and removed. In part (b)(ii), in a queue ADT, the node accessed for removal is always at the head of the queue, the candidate here wrongly uses CurrentPointer, which does not yet have a value. The candidate correctly updates HeadPointer by following the pointer of the current node, but does not appreciate that if the last name in the queue has just been removed the head pointer would now be null and therefore the tail pointer would also need to be set to null. The candidate does not demonstrate that linking the released node to the free list means linking it to the front of the list that FreePointer is pointing to, so the pointer of the released node and FreePointer need to be updated.

Marks awarded for part (a) = 1/3
Marks awarded for part (b) = (i) 3/4, (ii) 1/6

Total marks awarded = 5 out of 13

Cambridge International Examinations
1 Hills Road, Cambridge, CB1 2EU, United Kingdom
tel: +44 1223 553554 fax: +44 1223 553558
email: info@cie.org.uk www.cie.org.uk