CAMBRIDGE
International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9618/22**

Paper 2 Fundamental Problem-solving and Programming Skills **May/June 2025**

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 May/June 2025 series for most Cambridge IGCSE, Cambridge International A and AS Level components, and some Cambridge O Level components.

Due to a series-specific issue during the live exam series, all candidates were awarded full marks for questions 1(a)(ii) and 1(b)(ii). The mark scheme for these questions was not used by examiners.

This document consists of **20** printed pages.

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 descriptions 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.

Annotations guidance for centres

Examiners use a system of annotations as a shorthand for communicating their marking decisions to one another. Examiners are trained during the standardisation process on how and when to use annotations. The purpose of annotations is to inform the standardisation and monitoring processes and guide the supervising examiners when they are checking the work of examiners within their team. The meaning of annotations and how they are used is specific to each component and is understood by all examiners who mark the component.

We publish annotations in our mark schemes to help centres understand the annotations they may see on copies of scripts. Note that there may not be a direct correlation between the number of annotations on a script and the mark awarded. Similarly, the use of an annotation may not be an indication of the quality of the response.

The annotations listed below were available to examiners marking this component in this series.

Annotations

Annotation	Meaning
BOD	Benefit of the doubt
^	To indicate where a key word/phrase/code is missing
✗	Incorrect
FT	Follow through
〰	Indicate a point in an answer
Highlighted text	To draw attention to a particular aspect or to indicate where parts of an answer have been combined
I	Ignore
NAQ	Not answered question
NBOD	No benefit of doubt given
NE	No examples or not enough
⌇	Not relevant or used to separate parts of an answer
Off-page comment	Allows comments to be entered at the bottom of the RM marking window and then displayed when the associated question item is navigated to.
REP	Repetition

Annotation	Meaning
SEEN	Indicates that work or a page has been seen including blank answer spaces and blank pages.
✓	Correct
TV	Too vague

Mark scheme abbreviations

/	separates alternative words / phrases within a marking point
//	separates alternative answers within a marking point
Underline	actual word given must be used by candidate (grammatical variants accepted)
Max	indicates the maximum number of marks that can be awarded
( )	the word / phrase in brackets is not required, but sets the context
bold	word/phrase in bold indicates this is a key word/phrase in the candidates answer and this word/phrase or a word/phrase with a similar meaning must be present

Question	Answer	Marks
1(a)(i)	One mark for Choice of program development cycle depends on the **approach/method** required to produce the program **Or** One mark for a factor that would influence the choice of program development cycle, e.g. • Rapid development of program/prototypes required • Need for prototypes (at an early stage) • Complexity of problem • Skills / experience of development team / programmer(s) • Budget / Time / Resources available • Size of development team • Developer / programmer can return to earlier **stages** / any stage • Avoidance of repeating previous stages in development of program • How much involvement clients will have • Allows the (program) requirements to be changed during program development • (Program) requirements agreed at start of development of program **Max 1**	1
1(a)(ii)	Which programming language would best be suited to control the production line // Which programming language would best suit the problem being solved	1
1(b)(i)	Examples include: • Change to (production line) requirements • New technology / hardware available (to control production line) • Changes made to library modules **used** • Change in relevant legislation **Max 3**	3
1(b)(ii)	Perfective // Corrective	1
1(c)	1 mark for each correct row	4
1(d)(i)	Two / 2	1

1(c):

Expression	Data type
`RIGHT(MachineCode, 4)`	**STRING**
`Speed * 2.5`	**REAL**
`NOT Status`	**BOOLEAN**
`IS_NUM(Check)`	**BOOLEAN**

Question	Answer	Marks
1(d)(ii)	1000	**1**
1(d)(iii)	`DECLARE Product : ARRAY [0:99, 0:9] OF INTEGER` 1 mark for correct upper and lower bound • [0:99, 0:9 1 mark for all other parts of declaration `DECLARE Product : ARRAY [0:99, 0:9] OF INTEGER`	**2**

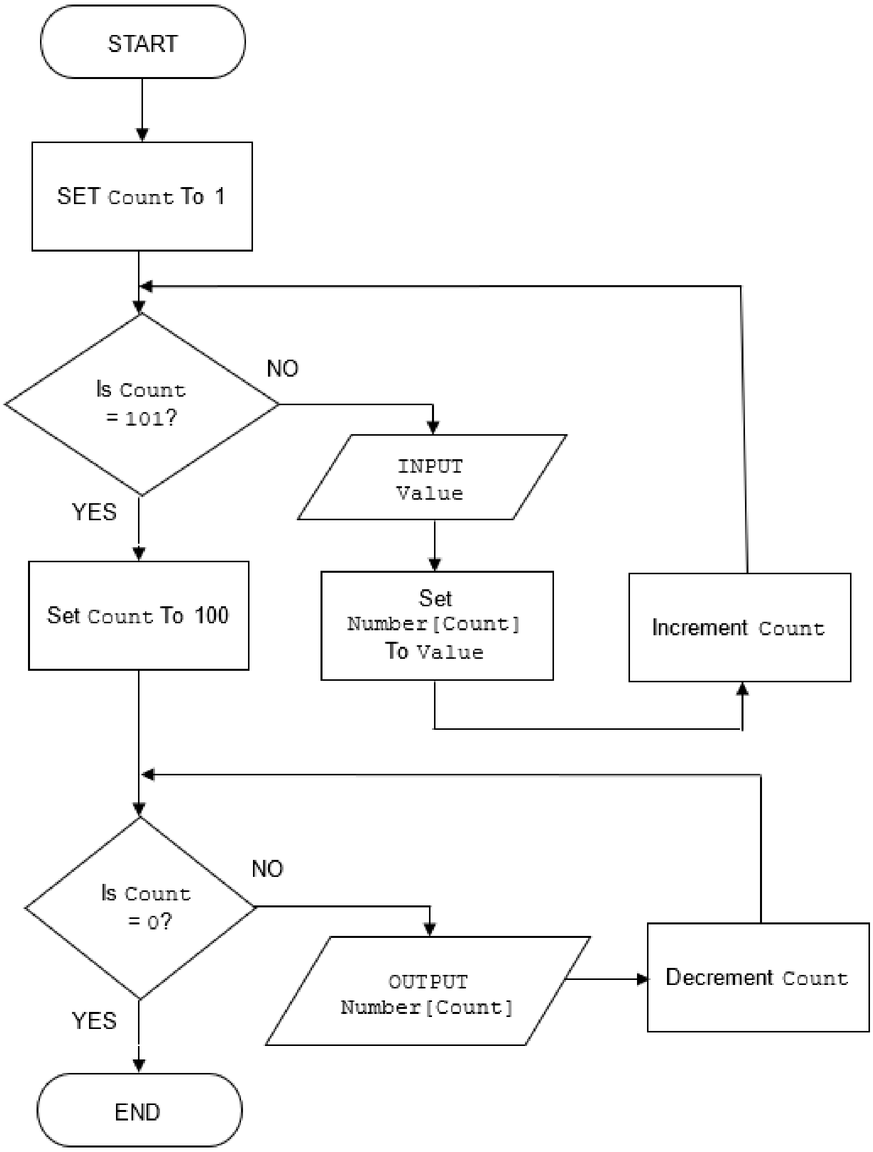
Question	Answer	Marks
2(a)	Mark as follows: 1. To increase the level of detail of the algorithm // To break the problem / task into **smaller steps** … 2. … until steps can be directly translated into lines of code // from which it can be programmed	**2**
2(b)(i)	One mark for each of set test values Hours worked between 1 and 40 inclusive Sales value <= 2000 Bonus Pay: 0 Hours worked above 40 Sales value <= 2000 Bonus Pay: 10 Hours worked between 1 and 40 inclusive Sales value > 2000 Bonus Pay: 50 Hours worked above 40 Sales value above 2000 Bonus Pay: 100 **max 2**	**2**
2(b)(ii)	One mark per point • Simple **modules** are written to **replace** each of the unfinished modules. • Each simple module will return an expected value / will output a message to show it has been called.	**2**
2(b)(iii)	One mark for naming type of error and one mark for corresponding description Type of error: Logic (error) Description: Where the program does not behave as expected / Does not give expected result / An error in the logic of the algorithm **Or** Type of error: Run-time (error) Description: The program performs an illegal operation **Max 2**	**2**

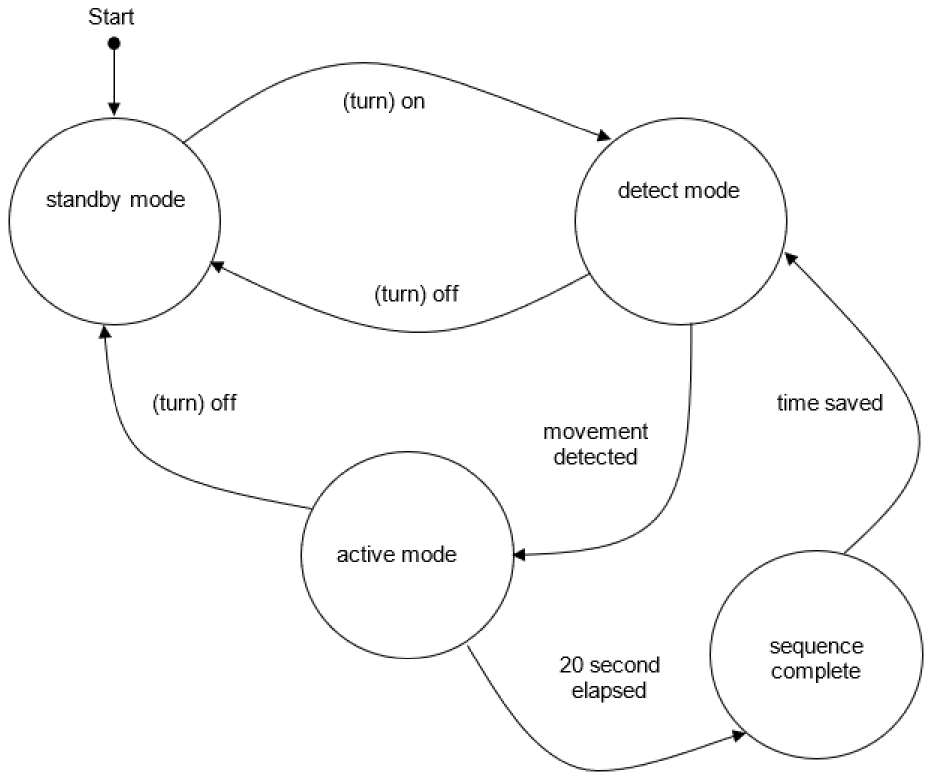
Question	Answer	Marks
3	Example solution	8

```
DECLARE FirstNumber : INTEGER
DECLARE SecondNumber : INTEGER
DECLARE ThirdNumber : INTEGER

FirstNumber ← INT(RAND(21)) - 10

OUTPUT FirstNumber

REPEAT
   SecondNumber ← INT(RAND(21)) - 10
UNTIL FirstNumber <> SecondNumber

OUTPUT SecondNumber

IF FirstNumber < 0 AND SecondNumber < 0 THEN
   ThirdNumber ← INT(RAND(6)) + 30
   OUTPUT ThirdNumber
ENDIF
```

Mark points:

1. Declare all variables used
2. Use `RAND()` function with any **integer** parameter
3. Use `INT()` function with any **numeric** parameter
4. Conditional loop until **two different** random numbers are generated
5. Use `INT()` function using random number generated between –10 and 10 inclusive **in a loop**
6. Output **two different** random integers / numbers
7. Check if both random integers / numbers are negative
8. then output a (third) random integer / number between 30 and 35 inclusive

Question	Answer	Marks
4	 START SET Count To 1 Is Count = 101? NO YES INPUT Value Set Number[Count] To Value Increment Count Set Count To 100 Is Count = 0? NO YES OUTPUT Number[Count] Decrement Count END One mark for each functional group as listed: 1. Initialise Index used for `Number` to either 1 or 0 before first loop 2. Loop 100 times 3. Input value 4. Increment array index **and** store value in `Number` array at element referenced by array `index` **in a loop** 5. Output loop starts at last element in `Number` array 6. Output `Number` array element referenced by `Index` **in a loop** 7. Output all elements of `Number` array **once** in reverse order **Max 6**	6

Question	Answer	Marks
5(a)	 Start standby mode (turn) on detect mode (turn) off (turn) off movement detected time saved active mode 20 second elapsed sequence complete Mark as follows: 1 Mark: The three new states drawn and labelled. 1 Mark: Two of the events drawn with labels connecting correct states and correct line direction 1 Mark: Four of the events drawn with labels connecting correct states and correct line direction. 1 Mark: All and only six events drawn with labels connecting correct states and correct line direction.	4

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>5(b)</td><td>

Conditional Solution

Example Solution

```
DECLARE Count : INTEGER
DECLARE Line : STRING
DECLARE NextHour, Hour : STRING

OPENFILE "TimeTaken.txt" FOR READ
Count ← 1
READFILE "TimeTaken.txt", Line
Hour ← LEFT(Line, 2)

WHILE NOT EOF("TimeTaken.txt")
   READFILE "TimeTaken.txt", Line
   NextHour ← LEFT(Line, 2)
   IF NextHour = Hour THEN
     Count ← Count + 1
   ELSE
      OUTPUT "Hour : ", Hour, " Total : ", Count
      Hour ← NextHour
      Count ← 1
   ENDIF
ENDWHILE

OUTPUT "Hour : ", Hour, " Total : ", Count

CLOSEFILE "TimeTaken.txt"
```

Mark as follows:

1. Any Initialisation of count for number of pictures taken each hour
2. Open "TimeTaken.txt" for read and subsequently close
3. Conditional loop until EOF
4. Read a line from "TimeTaken.txt" **in a loop**
5. Extract Hour from line read **in a loop**
6. A mechanism to compare current hour extracted from file with last one read from file **in a loop**
7. If hours same increment count **in a loop**
8. If **not** same output count (with a suitable message) **and** update Hour ← NextHour **and** set Count to 1 **in a loop**
9. Output final Count **and** Hour (with a suitable message) **once only**

Note: max 8

</td><td>8</td></tr>
</table>

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>5(b)</td><td>

Alternative solution and mark scheme use of an array to hold count for each hour
Also, for use of 24 variables and 24 selection conditions

```
DECLARE HoursArray[0 : 23] OF INTEGER
DECLARE Index, Hour : INTEGER
DECLARE Line : STRING

FOR Index ← 0 TO 23
   HoursArray[Index] ← 0
NEXT Index

OPENFILE "TimeTaken.txt" FOR READ

WHILE NOT EOF("TimeTaken.txt")
   READFILE "TimeTaken.txt", Line
   Hour ← STR_TO_NUM(LEFT(Line, 2))
   HoursArray[Hour] ← HoursArray[Hour] + 1
ENDWHILE

FOR Index ← 0 TO 23
   IF HoursArray[Index] <> 0 THEN
      OUTPUT "Hour : ", Index, " Total : ",
                                        HoursArray[Index]
   ENDIF
NEXT Index

CLOSEFILE "TimeTaken.txt"
```

Mark as follows:

1. Initialisation of 24 array elements to 0 // Initialisation of 24 Integer variables to 0
2. Open `"TimeTaken.txt"` for read and subsequently close
3. Conditional loop until EOF
4. Read a line from `"TimeTaken.txt"` **in a loop**
5. Extract Hour from line read **in a loop**
6. Convert Hour to an integer value **in a loop**
7. Increment appropriate array element / variable **in a loop**
8. Output of each hour and corresponding count variable (with a suitable message) for all values **where count is not zero**

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

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>6(a)(i)</td><td>
<table>
<tr><th>Value</th><th>Start</th><th>Unused</th><th>New</th><th>Last</th><th>Current</th><th></th></tr>
<tr><td>1043</td><td>2</td><td>8</td><td>8</td><td>-1</td><td>2</td><td>Region 1</td></tr>
<tr><td></td><td></td><td></td><td></td><td>2</td><td>3</td><td>Region 2</td></tr>
<tr><td></td><td></td><td></td><td></td><td>3</td><td>1</td><td>Region 3</td></tr>
<tr><td></td><td></td><td></td><td></td><td>1</td><td>7</td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td>7</td><td>4</td><td>Region 4</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
Award 1 mark per region
</td><td>4</td></tr>
<tr><td>6(a)(ii)</td><td>
<table>
<tr><th></th><th>Data</th></tr>
<tr><td>1</td><td>1018</td></tr>
<tr><td>2</td><td>1007</td></tr>
<tr><td>3</td><td>1010</td></tr>
<tr><td>4</td><td>1056</td></tr>
<tr><td>5</td><td>1092</td></tr>
<tr><td>6</td><td>1062</td></tr>
<tr><td>7</td><td>1034</td></tr>
<tr><td>8</td><td><b>1043</b></td></tr>
<tr><td>9</td><td>0</td></tr>
<tr><td>10</td><td>0</td></tr>
</table>
<table>
<tr><th></th><th>Pointer</th></tr>
<tr><td>1</td><td><b>7</b></td></tr>
<tr><td>2</td><td><b>3</b></td></tr>
<tr><td>3</td><td><b>1</b></td></tr>
<tr><td>4</td><td><b>6</b></td></tr>
<tr><td>5</td><td><b>-1</b></td></tr>
<tr><td>6</td><td><b>5</b></td></tr>
<tr><td>7</td><td><b>8</b></td></tr>
<tr><td>8</td><td><b>4</b></td></tr>
<tr><td>9</td><td><b>10</b></td></tr>
<tr><td>10</td><td><b>-1</b></td></tr>
</table>
Mark as follows:
1 mark for global array <code>Data</code> row 8 containing the value 1043
1 mark for global array <code>Pointer</code> row 7 containing the value 8 and row 8 containing the value 4
1 mark for all other rows in both arrays
</td><td>3</td></tr>
</table>

Question	Answer	Marks
6(b)	Example answer: The ADT is a linked list and the procedure `Place()` inserts / add a new node / value into it / the linked list Mark as follows: 1. 1 mark for identifying the ADT as a linked list 2. 1 mark for identifying operation as inserting / adding a value / node **into a** linked list	**2**

Question	Answer	Marks
7(a)	Example solution Conditional Loop	8

```
FUNCTION FindCustomer(CustomerID : INTEGER) RETURNS
INTEGER

   DECLARE Index : INTEGER
   DECLARE Found : BOOLEAN
   CONSTANT Unused = 99999
   CONSTANT Upper = 1000
   Found ← FALSE

   Index ← 1

   IF CustomerID < 10001 OR CustomerID > 11000 THEN
      RETURN -1  // Out of range value for customer ID
   ENDIF

   WHILE Found = FALSE AND Loyalty[Index, 1] <> 99999
      IF Loyalty[Index,1] = CustomerID THEN
         Found ← TRUE
      ELSE
         Index ← Index + 1
      ENDIF
      IF Index > Upper THEN
         RETURN -1
      ENDIF
   ENDWHILE

   IF Found THEN
      RETURN Loyalty[Index, 2]
   ELSE
      RETURN -1
   ENDIF

ENDFUNCTION
```

Mark as follows:

1. Create function header and ending with correct parameter and return type
2. Check `CustomerID` is in range and if not return –1
3. (Conditional) loop iterating through each element in array
4. Check for current element in array contains required customer ID **in a loop**
5. … If found set value to terminate loop
6. Terminating loop when customer ID found
7. Terminating loop when current customer ID is 99999
8. Return either loyalty points for the customer found or `-1` if not found

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>7(a)</td><td>

Alternative solution using FOR Loop

Example solution

```
FUNCTION FindCustomer(CustomerID : INTEGER) RETURNS INTEGER

   DECLARE Index : INTEGER

   IF CustomerID < 10001 OR CustomerID > 11000 THEN
      RETURN -1  // Out of range value for customer ID
   ENDIF

   FOR Index ← 1 TO 1000
      IF Loyalty[Index, 1] = CustomerID THEN
         RETURN Loyalty[Index, 2]
      ENDIF
      IF Loyalty[Index, 1] = 99999 THEN
         RETURN -1
      ENDIF
   NEXT Index

ENDFUNCTION
```

Mark as follows:

1. Create function header and ending with correct parameter and return type
2. Check `CustomerID` is within range and if not return –1
3. FOR Loop
4. Loop for elements 1 to 1000 / 0 to 999
5. Check if current element in array contains required Customer ID **in a loop**
6. … If found correctly return loyalty points // store correct loyalty points and return after loop
7. Check if current element in array is `99999` and return `-1` / break loop **in a loop**
8. return –1 if Customer ID not found

Direct access solution
Alternative solution subtracting 10000 from `CustomerID`

```
FUNCTION FindCustomer(CustomerID : INTEGER) RETURNS INTEGER

   DECLARE Index : INTEGER

   IF CustomerID < 10001 OR CustomerID > 11000 THEN
      RETURN -1  // Out of range value for customer ID
   ENDIF
```

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

Question	Answer	Marks
7(a)		

```
   Index = CustomerID - 10000

   IF Loyalty[Index, 1] = CustomerID THEN
      RETURN Loyalty[Index, 2]
   ENDIF

   RETURN -1


ENDFUNCTION
```

Mark as follows:

1. Create function header and ending with correct parameter and return type
2. Declare Index as Integer
3. Check `CustomerID` is less than `10001` and return `-1` if it is
4. Check `CustomerID` is greater than `11000` and return `-1` if it is
5. Calculate Index by subtracting a `10000` form `CustomerID`
6. Check if array contains the `CusomerID`
7. Return Loyalty Points if `CustomerID` found
8. Return `-1` if not found

Binary Search Mark Scheme

Example Solution

```
FUNCTION FindCustomer(CustomerID : INTEGER) RETURNS
                                                    INTEGER
   DECLARE Start : INTEGER
   DECLARE End : INTEGER
   DECLARE Mid : INTEGER

   IF CustomerID < 10001 OR CustomerID > 11000 THEN
      RETURN -1  // Out of range value for customer ID
   ENDIF

   Start ← 1
   End ←1000

   WHILE Start <= End
      Mid ← (Start + End) DIV 2
      IF Loyalty[Mid, 1] = CustomerID THEN
         RETURN Loyalty[Mid, 2]
      ELSE IF Loyalty[Mid, 1] > CustomerID THEN
         End ← Mid - 1
      ELSE
         Start ← Mid + 1
      ENDIF
   ENDWHILE
   RETURN -1
ENDFUNCTION
```

Question	Answer	Marks
7(a)	Mark points 1. Create function header and ending with correct parameter and return type 2. Check `CustomerID` is within range and if not return –1 3. Conditional loop that halves array search space with each iteration 4. Check if `CustomerID` is stored in current array element **in loop** 5. Mechanism to end loop if `CustomerID` is found in array **in a loop** 6. Mechanism to end loop if `CustomerID` is not in array **in a loop** 7. If `CustomerID` found in array then return correct loyalty points 8. If `CustomerID` not in array then return –1	

7(b)	Example solution
	
	`PROCEDURE PointsReport()`
	
	`   DECLARE Count : INTEGER`
	`   DECLARE Index : INTEGER`
	`   DECLARE Sum : INTEGER`
	`   DECLARE Average : REAL`
	
	`   Index ← 1`
	`   Count ← 0`
	`   Sum ← 0`
	
	`   WHILE Loyalty[Index, 1] <> 99999 AND Index <= 1000`
	`     IF Loyalty[Index, 2] >= 11 THEN`
	`       OUTPUT Loyalty[Index, 1]`
	`     ENDIF`
	`     Count ← Count + 1`
	`     Sum ← Sum + Loyalty[Index, 2]`
	`     Index ← Index + 1`
	`   ENDWHILE`
	
	`   Average ← Sum / Count`
	`   OUTPUT "The average points of all the customers is ",`
	`                                                  Average`
	
	`ENDPROCEDURE`
	
	Mark as follows:
	
	1. Create procedure header and ending
	2. Declare and initialise `Index`, `Sum` and `Count`
	3. Loop through all elements in `Loyalty` array
	4. … or terminates when column1 of `Loyalty` array equals `99999` **in a loop**
	5. Check if loyalty points greater than or equal to `11` **in a loop**
	6. If true output customer ID
	7 Sum points **and** Increment `Count` **in a loop**
	8 Calculate **and** output average with an appropriate message
	
	Max 7
		7

Question	Answer	Marks
7(c)(i)	An array can only store data of **the same type** // An array cannot store data **of different types**	**1**
7(c)(ii)	1 mark for each point 1. a new (composite) data type / record is defined (that consists of both `INTEGER` and `STRING`) 2. an array based on this new type is **declared** Alternative 1 mark for each point 1. Converting loyalty points to string and concatenating / joining / append with Customer ID 2. Storing **concatenated** string in an **array of string**	**2**