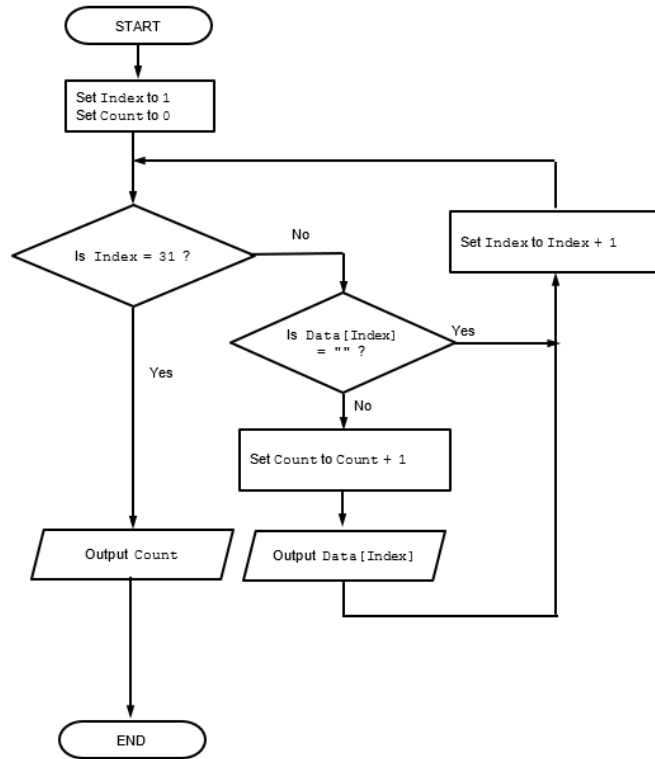


2

Example solution



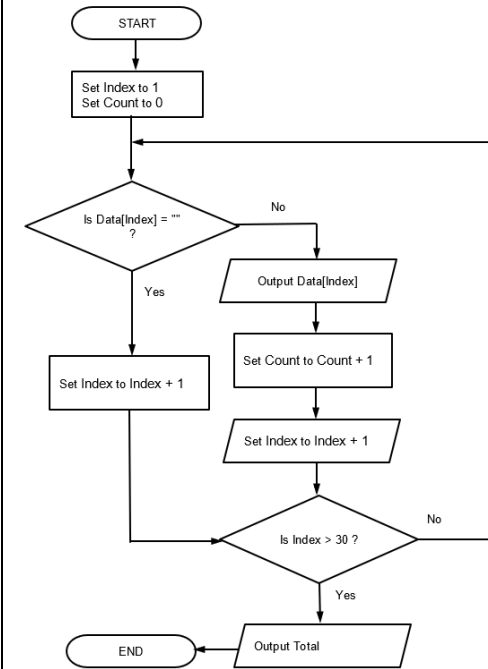
Mark as follows:

- 1 Initialisation of variables used as **array index** and **count**
- 2 Loop through exactly 30 elements in the array
- 3 Test for empty string in current array element // Test for non-empty string
- 4 Increment count if appropriate **in a loop**
- 5 Both required outputs under correct conditions with correct outputs

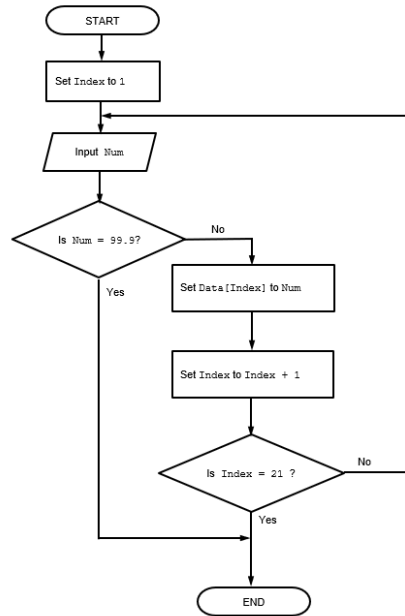
2

Alternative solution

Checking for empty string first



2

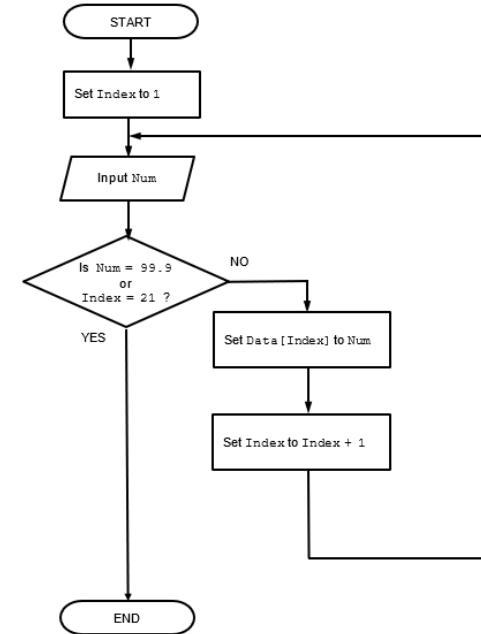


Mark as follows:

- MP1** Initialisation of the array index
MP2 Input number **in a loop**
MP3 Test for termination value (99.9) **and** correct ending
MP4 Test for array full / maximum 20 iterations **and** correct ending
MP5 Assign number to Data array element **and** increment index

2

Alternative solution:



2(a)

One mark per point:

- 1 Open the file in append mode (and subsequently close) **before loop**
- 2 Initialise a variable to 1 / 0 **before loop**
- 3 Use the variable as an index to the array to access an element **in a loop**
- 4 Test if element value is blank / empty **in a loop**
- 5 If not blank then write the value to the file **in a loop**
- 6 Increment the variable **in a loop**
- 7 Repeat from MP3 until variable value is 66 / 65 / all elements (in the array) have been checked

Max 6

2(b)

One mark per point:

Construct: Selection

Use: To test whether the (array) element is blank

Alternative:

Construct: Sequence

Use: To specify the order in which the steps of the algorithm have to be followed so that the task is completed correctly

3	Example solution: <pre> DECLARE GeneratedValue : INTEGER DECLARE Guess : INTEGER GeneratedValue ← INT(RAND(100)) + 1 REPEAT OUTPUT "Enter a guess between 1 and 100: " INPUT Guess IF Guess > GeneratedValue THEN OUTPUT "Guess was too high" ENDIF IF Guess < GeneratedValue THEN OUTPUT "Guess was too low" ENDIF UNTIL Guess = GeneratedValue OUTPUT "Number guessed correctly" MP1 Declare all variables used MP2 Uses RAND() function MP3 Uses INT() function MP4 Correct syntax for random number between 1 and 100 MP5 Conditional loop until correct number guessed MP6 Prompt and input a guess in a loop MP7 Check if guess is too high in a loop MP8 Check if guess is too low in a loop MP9 Output appropriate message (x2) when the number is not guessed correctly (in the loop) and output correct guess message Max 8 </pre>
---	---

2(a)	Mark as follows: - To increase the level of detail of the algorithm // To break the problem / task into smaller steps until steps can be directly translated into lines of code // from which it can be programmed
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(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(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

3

Example solution

```

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

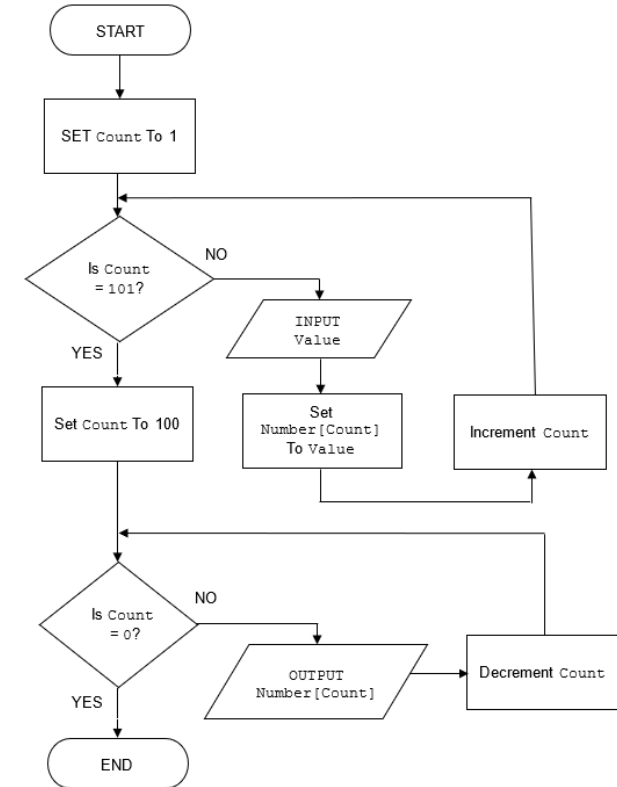
IF FirstNumber < 0 AND SecondNumber < 0 THEN
    ThirdNumber ← INT(RAND(6)) + 30
    OUTPUT ThirdNumber
ENDIF

```

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

4

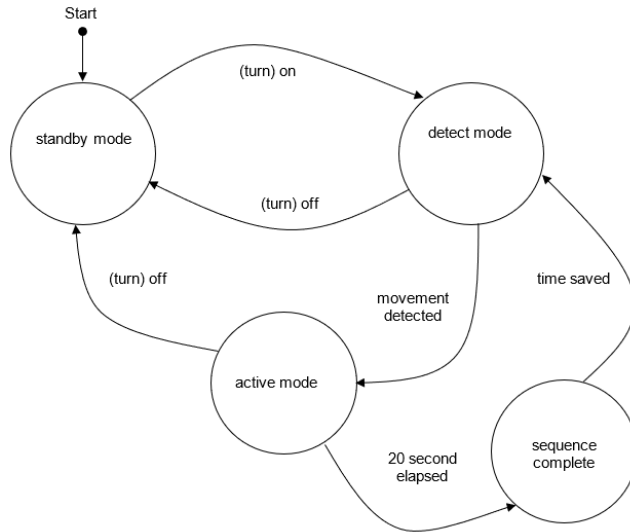


One mark for each functional group as listed:

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

5(a)



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.

5(b)

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:

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

Note: max 8

5(b)	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 <pre> 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" </pre> 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
------	---

4(a)	MP1 Open the file (Sales.txt) in read mode and subsequently close MP2 Read a line (from the text file) MP3 Convert the line read (string) to a number MP4 If the number is greater than 500 and then output week number MP5 Increment week number (must have been Initialised ...) MP6 Repeat from step 2 for 52 times // Repeat from step 2 until end of file Max 5
------	---

4(b)	MP1 Going through the program a line / statement at a time // doing a dry run // single-stepping (at run-time with an IDE) // Peer testing/review is carried out MP2 Creating a trace table MP3 Draw up test data // draw up list of inputs with expected output // boundary, normal, abnormal data is used MP4 Errors will be indicated when a variable / output gives an unexpected value MP5 Faults in the logic of the program can be detected // Errors may be indicated by an unexpected path through the program Max 3
------	---

2(a)	<pre> DECLARE Count, Total, NextNumber : INTEGER Total ← 0 FOR Count ← 1 TO 100 OUTPUT "Input an integer value" INPUT NextNumber IF NextNumber > 0 THEN Total ← Total + NextNumber ENDIF NEXT Count OUTPUT Total </pre> Mark as follows: MP1 Declarations of all variables used MP2 Loop for 100 iterations MP3 Prompt and input a value in a loop and MP4 Test for value > 0 // >=1 in a loop MP5 Sum the Total in a loop MP6 Output of Total after the loop Max 5 marks
2(b)	MP1 Construct: Iteration / Repetition MP2 Use: To loop through all 100 inputs // To loop 100 times ALTERNATIVE: MP1 Construct: Selection MP2 Use: To test whether the value input is positive

4(a)	One mark per highlighted part: <pre> FOR Index ← 1 TO 5 Lower ← GB[Index] - 2 Upper ← <u>GB[Index] + 2</u> // <u>Lower + 4</u> IF Mark <u>>= Lower</u> AND Mark <u><= Upper</u> THEN //IF Mark <u><= Upper</u> AND Mark <u>>= Lower</u> THEN OUTPUT "Check this paper" ENDIF NEXT Index </pre>
4(b)(i)	MP1 Use Mark as the <u>index</u> to the Check array // to specify an array element MP2 If value of indexed element is TRUE, then the paper will need to be checked

4(b)(ii)	Example:solution <pre> PROCEDURE GBInitialise() DECLARE GBIndex, GBMark, ThisIndex : INTEGER FOR ThisIndex ← 0 TO 75 Check[ThisIndex] ← FALSE // initialise all values NEXT ThisIndex FOR GBIndex ← 1 TO 5 GBMark ← GB[GBIndex] FOR ThisIndex ← GBMark - 2 TO GBMark + 2 Check[ThisIndex] ← TRUE //Set GBMark ± 2 to TRUE NEXT ThisIndex NEXT GBIndex ENDPROCEDURE </pre> Mark as follows: MP1 Procedure heading and ending MP2 Initialise Check array to FALSE MP3 Loop through GB array MP4 Extract the GB mark MP5 Loop for 5 elements across GBMark ± 2 // Check if within GBMark ± 2 MP6 Assign each element of Check array to TRUE ALTERNATIVE – Example using selection Version 1 <pre> FOR ThisIndex ← 0 TO 75 CASE ThisIndex GB[1] - 2 TO GB[1] + 2 : Check[ThisIndex] ← TRUE GB[2] - 2 TO GB[2] + 2 : Check[ThisIndex] ← TRUE GB[3] - 2 TO GB[3] + 2 : Check[ThisIndex] ← TRUE GB[4] - 2 TO GB[4] + 2 : Check[ThisIndex] ← TRUE GB[5] - 2 TO GB[5] + 2 : Check[ThisIndex] ← TRUE OTHERWISE: Check[ThisIndex] ← FALSE ENDCASE NEXT ThisIndex </pre>
----------	---

4(b)(ii)	ALTERNATIVE – Example using selection Version 2 <pre> DECLARE ThisIndex, GBIndex : INTEGER DECLARE Lower, Higher : INTEGER FOR ThisIndex ← 0 TO 75 For GBIndex ← 1 TO 5 Lower ← GB[GBIndex] - 2 Higher ← GB[GBIndex] + 2 IF ThisIndex >= Lower AND ThisIndex <= Higher THEN Check[ThisIndex] ← TRUE ELSE Check[ThisIndex] ← FALSE ENDIF NEXT Index NEXT ThisIndex </pre> Mark as follows: MP1 Procedure heading and ending MP2 Loop through <i>Check</i> array// loop from 0 to 75 MP3 Attempt at using CASE / Selection structure with five clauses/ five checks for range MP4 Extract GB grade MP5 Compare <i>ThisIndex</i> with each element in GB array ±2 MP6 Assign each element of <i>Check</i> array to TRUE if in range or FALSE if not in range
----------	--

2(a)	<pre> graph TD START([START]) --> Init[Set Tries to 0 Set Count to 0] Init --> LoopStart(()) LoopStart --> Input[/INPUT A, B/] Input --> IncTries[Set Tries to Tries + 1] IncTries --> IsAB[Is A > B?] IsAB -- Yes --> IncCount[Set Count to Count + 1] IsAB -- No --> IsCount10{Is Count = 10?} IncCount --> IsCount10 IsCount10 -- Yes --> Output[/OUTPUT Tries/] Output --> END([END]) IsCount10 -- No --> LoopStart </pre> One mark per outlined region: 1 Initialise both counts 2 Increment <i>Tries</i> every time a pair is input 3 Compare $A > B$ and increment <i>Count</i> if TRUE 4 Test for $\text{Count} = 10$ (10th time $A > B$) – MUST include Yes / No labels 5 If so output <i>Tries</i>, otherwise loop
2(b)	One mark per point: 1 A (variable of type) string will be input // by example e.g. "67,72" 2 A special / identified character would need to be used to separate each numeric value // all numbers are fixed length