1(a)

Expression
`MyString ← 'X' & MyString`
`MyChar ← MID ("ABCD", 1 / 2 / 3 / 4 , 1)`
`MyString ← NUM_TO_STR (DAY / MONTH / YEAR / DAYINDEX (MyDOB))`
`MyInt ← INT (LENGTH (MyString) / 2)`

One mark per row

1(b)

Test description	Test method
carried out as soon as a program module has been coded	alpha
carried out as program modules are being combined	**integration**
completed by the developers without referring to the code	**black-box**
completed by the customer	**acceptance** / beta

One mark per row (2 to 4)

1(c)

One mark per point:

1 reference to a **breakpoint**
2 reference to **single stepping**
3 Correct explanation of **both** e.g. to stop program execution at specific point / line / statement / instruction and to execute one line / statement / instruction at a time to (locate an/the error)

One mark for each bulleted point

1(a)

Pseudocode example	Selection	Iteration	Subroutine

`IF Status = FALSE THEN`
`  FOR Count ← 1 TO 20`
`  CALL Reset(Count)`
`  NEXT Count`
`ENDIF`
	✓	✓	✓

`OTHERWISE : Status ← TRUE`	✓		
`WHILE AllDone() = TRUE`		✓	✓
`30 : NextChar ← 'X'`	✓		

One mark per row

1(b)

Variable	Example data value	Data type
`Result`	`5.42`	**REAL**
`MonthLetter`	`"JFMAMJJASOND"`	**STRING**
`Birthday`	`15/11/2009`	**DATE**

One mark per row

1(c)

Expression	Evaluates to
`INT(Result) + 1 > 6`	**FALSE**
`NUM_TO_STR(LENGTH(MonthLetter))`	**"12"**
`NUM_TO_STR(Result + "3.2")`	**ERROR**
`MID(MonthLetter, MONTH(Birthday) – 2, 1)`	**"S"**

One mark per row

2(a)(i)	**MP1** The two decision boxes are in the wrong order // The **first decision** symbol is wrong **MP2** The first decision should check for values greater than or equal to 2000 **Max 1**
2(a)(ii)	Explaining how to correct the flowchart. **MP1** Swap around the two decision boxes **MP2** The first decision box should check for values greater than or equal to 2000 **and** the second decision box should check for values above 4000 **Max 1**
2(b)(i)	2000 / 4000 / 10 / 100
2(b)(ii)	**MP1** The value **cannot be changed** // avoids being different in two places **MP2** Makes the program easier to understand **MP3** Avoids repeatedly writing the same value throughout the program **MP4** A change to the value requires a change in only one statement **Max 2**
2(b)(iii)	**MP1** (The code is) tried and tested so error free **MP2** Provides functionality / code that the programmer may find difficult to code themselves **Max 1**
2(c)	**MP1** **Syntax (error)** **MP2** Rules of programming language // the language grammar was not followed **MP3** **Run-time (error)** **MP4** The program performs an illegal operation Mark as follows: One mark for naming each type of error **and** one mark for the description

1(a)	One mark per point: Error: The **calculation** of TaxPayable is incorrect Correction: TaxPayable ← (ItemCost * TaxRate) / 100
1(b)(i)	Use constants (to represent the tax rate values)
1(b)(ii)	One mark per bullet point (or equivalent to max 3): 1 **Tax rates** are entered once only 2 Avoids / Minimise (input) error(s) / changing the Tax rates accidentally // avoids different values for tax rates at different points in the program 3 When required, the constant value (representing a tax rate) is changed (once) // Easier to maintain / update the program (when the tax rates change) 4 Makes the program / code easier to understand
1(c)	One mark per row: **Variable name** | **Data type** HighRate | Boolean TaxPayable | Real
1(d)	OTHERWISE

1(a)

Pseudocode example	Selection	Iteration	Subroutine

`FOR Index ← 1 TO 3`
`  IF Safe[Index] = TRUE THEN`
`  Flap[Index] ← 0`
`  ENDIF`
`NEXT Index`
	✓	✓	

`CASE OF Compound(3)`	✓		✓
`REPEAT UNTIL AllDone() = TRUE`		✓	✓
`WHILE Result[3] <> FALSE`		✓	

One mark per row

1(b)

Variable	Example data value	Data type
`Available`	`TRUE`	BOOLEAN
`Received`	`"18/04/2021"`	STRING
`Index`	`100`	INTEGER

One mark per row

1(c)

Expression	Evaluates to
`Available AND NOT(Index > 100)`	TRUE
`Index MOD 30`	10
`NUM_TO_STR(Index + "33")`	ERROR

One mark per row

6

Loop Solution
Example solution:

```
FUNCTION AdjustClock(ThisYear : INTEGER) RETURNS INTEGER
   DECLARE ThisDayNumber, SundayCount : INTEGER
   DECLARE ThisDate : DATE

   ThisDayNumber ← 0
   SundayCount ← 0
   REPEAT
      ThisDayNumber ← ThisDayNumber + 1
      ThisDate ← SETDATE(ThisDayNumber, 3, ThisYear)
      IF DAYINDEX(ThisDate) = 1 THEN
         SundayCount ← SundayCount + 1
      ENDIF
   UNTIL SundayCount = 3
   RETURN ThisDayNumber
ENDFUNCTION
```

Mark as follows:
MP1 Function heading, parameter, ending and return type
MP2 Declare local integer variable that is used to create a date
MP3 Loop until 3rd Sunday found
MP4 Attempt to use both `SETDATE()` **and** `DAYINDEX()` **in a loop**
MP5 Correctly generate value of type `DATE` using `SETDATE()` **in a loop**
MP6 Test if value represents a Sunday using `DAYINDEX()` **in a loop**
MP7 Increment Sunday count **in a loop and** initialised correctly before loop
MP8 Return day number of **third** Sunday

Max 7 marks

Non-loop solution
Example solution:

```
FUNCTION AdjustClock(ThisYear : INTEGER) RETURNS INTEGER
   DECLARE FirstDayIndex: INTEGER
   DECLARE ThisDate : DATE

   ThisDate ← SETDATE(1, 3, ThisYear)
   FirstDayIndex ← DAYINDEX(ThisDate)
   IF FirstDayIndex = 1 THEN
      ThirdSunday ← FirstDayIndex + 14 //First day is
                                                    Sunday
   ELSE
      ThirdSunday ← 23 - FirstDayIndex // Other days
   ENDIF
   RETURN ThirdSunday
ENDFUNCTION
```

6	Mark as follows: **MP1** Function heading, parameter, ending and return type **MP2** Declare local integer variable that is used to hold a day number **MP4** Attempt to use both `SETDATE()` **and** `DAYINDEX()` **MP5** Correctly generate value of type `DATE` using `SETDATE()` for first of March / specific day in March **MP6** Test if date represents a Sunday / specific day using `DAYINDEX()` and calculate third Sunday // Correctly calculate third Sunday for a value returned by `DAYINDEX()` for one day **MP7** Calculate third Sunday for other six days **MP8** Return day number of **third** Sunday **Max 7 marks**

1(a)

Assignment statement	Data type
`A ← LEFT(MyName, 1)`	CHAR / STRING
`B ← Total * 2`	INTEGER / REAL
`C ← INT(ItemCost) / 3`	REAL
`D ← "Odd OR Even"`	STRING

One mark per row

1(b)

Expression	Evaluates to
`Tries < 10 AND NOT Sorted`	TRUE
`Tries MOD 4`	1
`TO_LOWER(MID(ID, 3, 1))`	'a' // "a"
`LENGTH(ID & "xx") >= Tries`	TRUE

One mark per row

1(c)(i)	The names do not reflect / indicate the purpose of the variable // the names are not meaningful
1(c)(ii)	They make the program more difficult to understand / debug / maintain
1(c)(iii)	One mark from: • Indentation / use of white space • Capitalisation of keywords • Use of comments • Use of modular programming • Use of local variables **Note: max 1 mark**

1(a)

Pseudocode example	Selection	Iteration	Input/Output

`FOR Index ← 1 TO 10`
`  Data[Index] ← 0`
`NEXT Index`
		✓	

`WRITEFILE ThisFile, "****"`			✓
`UNTIL Level > 25`		✓	

`IF Mark > 74 THEN`
`  READFILE OldFile, Data`
`ENDIF`
	✓		✓

One mark per row.

1(b)

Expression
`MyInt ← INT (3.1415926)`
`MyChar ← MID ("Elwood", 3, 1)`
Any of: • `MyString ← NUM_TO_STR (INT (27.509))` • `MyString ← CHR (INT (27.509))` • `MyString ← TO_UPPER( NUM_TO_STR( 27.509))` • `MyString ← TO_LOWER( NUM_TO_STR( 27.509))`
Any of: • `MyInt ← STR_TO_NUM ( RIGHT ("ABC123", 3))` • `MyInt ← LENGTH ( RIGHT ("ABC123", 3))` • `MyInt ← LENGTH ( LEFT ("ABC123", 3))`

One mark per row

1(c)

1 mark for stating a suitable way of documenting:
- Identifier table
  1 mark for giving one piece of information that should be recorded.
  examples include:
  Explanation of what (each) variable is used for
  The purpose of (each) variable
  An example of data values stored // Initialisation value