MAXIMUM MARK: 75
Dim HomeTeamName As String
Dim AwayTeamName As String
Dim WinningTeamName As String
Dim HomeRuns As Integer
Dim AwayRuns As Integer
Dim RunDifference As Integer

HomeTeamName = Console.ReadLine
HomeRuns = Console.ReadLine
AwayTeamName = Console.ReadLine
AwayRuns = Console.ReadLine

If HomeRuns > AwayRuns Then
    WinningTeamName = HomeTeamName
Else
    WinningTeamName = AwayTeamName
End If

RunDifference = Math.Abs(HomeRuns - AwayRuns)

Console.WriteLine("Winning team was " & WinningTeamName & " who scored " & RunDifference & " more runs")

Mark as follows:
Declaration of name strings [1]
Declaration of scores [1]
Input for name strings [1]
Input of two scores [1]
Calculation of the runs difference [1]
Calculation of the difference [1]
2 × IF or IF-THEN-ELSE used [1]
Stored as WinningTeamName [1]
Output shows team and runs difference [1]

[Total: 9]
2 (a) (i) **Identifier table:**

- INTEGER
  - Explanation – the next number selected

(ii) **Pseudocode:**

```plaintext
FOR Counter \( \leftarrow 1 \) to 6
  NextNumber \( \leftarrow \) INT(RND() \* 50) + 1
  OUTPUT NextNumber
ENDFOR / anything to mark the end of the loop
OUTPUT “That completes the draw”
```

(b) Program code demonstrates:

- declaration of variables
- correctly formed ‘count-controlled’ loop
- clear use of relevant inbuilt function

(c) (i) Explanation, e.g., It is not known how many times the loop needs to be executed to generate 6 different numbers.

(ii) any post-condition or pre-condition loop

(iii) **PROCEDURE** InitialiseNumberDrawn

```plaintext
FOR Index \( \leftarrow 1 \) TO 50
  NumberDrawn[Index] \( \leftarrow \) FALSE
ENDFOR
END PROCEDURE
```

(iv) **CALL** InitialiseNumberDrawn

```
Generated \( \leftarrow 0 \\
REPEAT // start of loop
  NextNumber \( \leftarrow \) GenerateNumber()
  IF NumberDrawn[NextNumber] = FALSE THEN
    OUTPUT NextNumber
    Generated \( \leftarrow \) Generated + 1
    NumberDrawn[NextNumber] \( \leftarrow \) TRUE
  ENDIF
UNTIL Generated = 6 // end of loop
OUTPUT “That completes the draw”
```
(v) NumberDrawn

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FALSE</td>
</tr>
<tr>
<td>2</td>
<td>FALSE</td>
</tr>
<tr>
<td>3</td>
<td>TRUE</td>
</tr>
<tr>
<td>4</td>
<td>FALSE</td>
</tr>
<tr>
<td>5</td>
<td>FALSE</td>
</tr>
<tr>
<td>6</td>
<td>FALSE</td>
</tr>
<tr>
<td>7</td>
<td>FALSE</td>
</tr>
<tr>
<td>8</td>
<td>FALSE</td>
</tr>
<tr>
<td>9</td>
<td>TRUE</td>
</tr>
<tr>
<td>10</td>
<td>FALSE</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>FALSE</td>
</tr>
<tr>
<td>40</td>
<td>FALSE</td>
</tr>
<tr>
<td>41</td>
<td>FALSE</td>
</tr>
<tr>
<td>42</td>
<td>TRUE</td>
</tr>
<tr>
<td>43</td>
<td>FALSE</td>
</tr>
<tr>
<td>44</td>
<td>FALSE</td>
</tr>
<tr>
<td>45</td>
<td>FALSE</td>
</tr>
<tr>
<td>46</td>
<td>FALSE</td>
</tr>
<tr>
<td>47</td>
<td>TRUE</td>
</tr>
<tr>
<td>48</td>
<td>FALSE</td>
</tr>
<tr>
<td>49</td>
<td>FALSE</td>
</tr>
<tr>
<td>50</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Mark as follows:
4 × correct ‘TRUE’ cells [1]
All other cells FALSE [1]
All cells contain something [1]

(vi) 3 47 9 42 [1]

[Total: 23]
3 (a) (i) 1 the identifier name for the function (chosen by the programmer) [1]
2 the parameter [1]
3 data type (for the parameter) [1]
4 data type for the value returned by the function [1]

(ii) Variable PossibleWinner stores the value returned by the function. [1]

(b) The data must be available each week.
When the program terminates after each weekly run, the data must be saved. [1]

(c) Labelled as follows:

 Museum 1
 READ
 PREVIOUSWINNERS.DAT
 data to array Winners

 Museum 2
 – Generate a member number
 – Decide whether this number is a new winner

 Museum 3
 FUNCTION GenerateNumber (NoOfMembers)

 Museum 4
 Search array Winners to confirm this is a new winner

 Museum 5
 – Search for
 ConfirmedWinningNumber
 in MEMBERS.DAT
 – RETURN MemberName

(d) (i) Index- INTEGER – Array subscript [3]
(ii) **Mark as follows:**

   - procedure header [1]
   - open the file [1]
   - correct open mode used [1]
   - index initialised [1]
   - loop [1]
   - read line of text [1]
   - assign to next array element [1]
   - increment index [1]
   - test for EOF [1]
   - output message shown [1]

   [max 8]

(e) (i) \( \text{DataLength} \leftarrow \text{LEN(MemberData)} \) [1]

(ii) \( \text{MemberNumber} \leftarrow \text{LEFT(MemberData, 4)} \) [1]

(iii) \( \text{MemberName} \leftarrow \text{MID(MemberData, 6, DataLength - 5)} \) [1]

[Total: 27]

4 (a) (i) \( P \) [1]

(ii) 87 [1]

(b) 84 [1]

(c) PEKOHOX [1]
(d) (i) INPUT MessageString
LengthMessageString ← LEN(MessageString)
NewString ← “”
FOR CharacterPosition ← 1 TO LengthMessageString
   Found ← FALSE
  /Index ← 1
   REPEAT
      IF MessageString[CharacterPosition] = Alphabet[Index]
         THEN
            SubstituteCharacter ← Substitute[Index]
            Found ← TRUE
      ELSE
         Index ← Index + 1
      ENDIF
   UNTIL Found
   NewString ← NewString + SubstituteCharacter
ENDFOR
OUTPUT NewString

Mark as follows:
input of the string
assign NewString as empty
calculation of the string length
outer loop
for ‘length’ iterations
compare individual characters with Alphabet array
inner loop to search for character
controlled with a counter
new substitute character added to NewString
final output of NewString

(ii) The code to search the Alphabet array can be avoided. / The ASCII codes for the letters are in sequence.

Example – index position for any character is ASC(<char>) − 64

[Total: 16]