Chapter 12

End of Chapter Exercises

1. *Given the following declaration:*

```
ARRAY:numbers OF [7] integer ;
```

And assuming the array has been populated with the values 2, 4, 6, 8, 10, 12, 14, what would the following `Display` statements output?

```
Display (numbers [4]) ;

Display (numbers [1]) ;

Display (numbers [6]) ;

Display (numbers [0]) ;

Display (numbers [7]) ;
```

10 – index 4 is the fifth element

4 – index 1 is the second element

14 – index 6 is the seventh element

2 – index

Error – index 7 doesn't exist; the array elements are numbered from 0 to 6.

2. *Given the following declaration:*

```
ARRAY:salesData OF [25] [12] real ;
```

How many rows does the array `salesData` have? How many columns does it have?

The array `salesData` has 25 rows and 12 columns.

3. *Using the array declaration from exercise 2 above, state which of the sox statements below, a, b, c, d, e, or f correctly places the value 100 into the array at row 10 column 6:*

a.
```
salesData [10] [6] ← 100 ;
```

b.
```
salesData [6] [10] ← 100 ;
```

c. `salesData [11] [7] ← 100 ;`

d. `salesData [7] [11] ← 100 ;`

e. `salesData [9] [5] ← 100 ;`

f. `salesData [5] [9] ← 100 ;`

Arrays use zero-based indexing, so the 10th row will be index 9 and the 6th column will be index 5. The row index always comes before the column index, so statement (e) is the one we want.

4. *Earlier we introduced the function `Length` which returns the size of an array expressed as the maximum number of its elements. A problem with this function is that it can cause the algorithm designer to inadvertently introduce a bug into their program because of zero-based array indexing. If an array has ten elements `Length` will return the value 10, but because of zero-based indexing, the 10th element of the array is accessed by an index value of 9, i.e. `Length (array)` – 1. Using the HTTLAP strategy design a new function `MaxIndex` that returns the index value of an array's final element rather than the array's length. That is, `MaxIndex` would return 9 for an array with a maximum of 10 elements, 99 for a 100-element array, and so on. Your solution can make use of the `Length` function.*

```
FUNCTION MaxIndex (ARRAY:theArray) RETURNS integer
   RETURN Length (theArray) -1 ;
ENDFUNCTION
```

5. *Design a program that solves the problem of allowing a user to create a shopping basket for an on-line store. The program should allow the user to enter products (up to a maximum of fifty items) which will be added to the basket (stored as an array). Do not worry about deletion at this time. Items will always be added to the end of the array.*

Let's say that products are identified by a unique integer product code. We could define the shopping basket thus:

```
ARRAY:basket OF [50] integer ;
```

Then we can fill the basket thus:

```
integer:count ,
        productId ;

count ← 0 ;
WHILE (user not finished) AND (count ≤ MaxIndex (basket))
```

```
   Select a product to buy ;
   basket [count] ← productId ;
   count ← count + 1 ;
ENDWHILE
```

6. *Extend your solution to task 2 to allow items to be deleted from the list. You need to consider what happens to gaps: should items be moved up the array to close haps left by deletion? Start by solving the simpler problem of deleting a value without considering the gaps. Once you are happy with that solution, then tackle the problem of the gaps. For example, if the second array element is deleted all the elements from the third downwards should be moved up one position.*

To delete one item without closing the gaps:

```
PROCEDURE DeleteItem (REFERENCE:ARRAY:theBasket, integer:index)
   theBasket [index] ← 0 ;
ENDPROCEDURE
```

But to close the gaps:

```
PROCEDURE DeleteItem (REFERENCE:ARRAY:theBasket, integer:index)
   integer:count ;
   FOR count GOES FROM index + 1 TO MaxIndex (theBasket)
      theBasket [index-1] ← theBasket [index] ;
      theBasket [index] ← 0 ;
   ENDFOR
ENDPROCEDURE
```

Note, the setting of each element to 0 after it has been copied could be avoided if we also keep track of the index of the current last item in the basket, with a variable called `lastIndex`, say. Then, we simply copy items back from `index` to `lastIndex` and then set the element at `lastIndex` to zero.

7. *The algorithm for randomly accessing the `cakeTally` array (see section on random access in arrays) could be written at least two other ways by using a continuation flag instead of a `cakeType` value greater than 2 being entered. Using the HTTLAP strategy design two alternative algorithms for this problem. One should a `WHILE` loop and the other a `DO…WHILE` loop.*

Here's the original algorithm:

```
ARRAY:cakeTally OF [3] integer ;
integer:counter ;
FOR counter GOES FROM 0 TO Length (cakeTally) - 1
   cakeTally [counter] ← 0 ;
ENDFOR
```

```
integer:cakeType,
        quantity ;
Get (keyboard, REFERENCE:cakeType, REFERENCE:quantity) ;
WHILE (cakeType ≤ Length (cakeTally) - 1)
   cakeTally [cakeType] ← cakeTally [cakeType]+ quantity ;
   Get (keyboard, REFERENCE:cakeType, REFERENCE:quantity) ;
ENDWHILE
```

Here's how we could rewrite it with a `WHILE` and a continuation flag

```
character:response ;
response ← 'Y' ;
WHILE (response= 'Y') OR (response = 'y')
   Get (keyboard, REFERENCE:cakeType, REFERENCE:quantity) ;
   cakeTally [cakeType] ← cakeTally [cakeType]+ quantity ;
   Display ('Another cake? Y/N') ;
   Get (keyboard, REFERENCE:response) ;
ENDWHILE
```

And with a `DO…WHILE`

```
character:response ;
DO
   Get (keyboard, REFERENCE:cakeType, REFERENCE:quantity) ;
   cakeTally [cakeType] ← cakeTally [cakeType]+ quantity ;
   Display ('Another cake? Y/N') ;
   Get (keyboard, REFERENCE:response) ;
WHILE (response= 'Y') OR (response = 'y')
```

8. *Look at the patient record structure in section 12.1. As it stands, it has fields to hold various address and date items. It might be more natural to think of an address as a record structure in its own right, and similarly a date. Amend the record type by creating two new types:* `AddressRecord` *and* `DateRecord`*. The address record structure will have the following fields:* `streetAddressLine1`, `streetAddressLine2`, `postalTown`, *and* `postalCode` *each of which should be a string. A* `DateRecord` *will have three integer fields:* `day`, `month`, *and* `year`*. Using these two new types, amend the* `PatientRecord` *type by removing the address and date fields and replacing them by three new fields:* `address` *(of type* `AddressRecord`*),* `birthDate`, *and* `joinDate` *(which should both be of type* `DateRecord`*).*

Patient records.

```
NEWTYPE AddressRecord IS
   RECORD
   string:streetAddressLine1,
```

```
            streetAddressLine2,
            postalTown,
            postalCode;
   ENDRECORD
NEWTYPE DateRecord IS
   RECORD
   integer:day,
           month,
           year;
   ENDRECORD
NEWTYPE PatientRecord IS
   RECORD
   string: title,
           familyName,
           givenName ;
   integer:birthday,
           birthMonth,
           birthYear ;
   AddressRecord:address ;
   string:telephone ;
   DateRecord:birthDate,
              joinDate ;
   ENDRECORD
```

9. *Change Solution 12.13 so that it displays the table in transposed form where the rows appear in reverse order.*

Here's solution 12.13:

```
FOR column GOES FROM 0 TO columns -1
   FOR row GOES FROM 0 TO rows -1
      Display (salesTable [row, column])  ;
   ENDFOR
   Display (↲) ;
ENDFOR
```

To display the rows in reverse order we simply need to make the inner `FOR` loop count down rather than up:

```
FOR column GOES FROM 0 TO columns -1
   FOR row GOES FROM rows -1 TO 0
      Display (salesTable [row, column])  ;
   ENDFOR
```

```
   Display (↲) ;
ENDFOR
```

10. *An input file, numbers.txt, contains 10 real numbers. Design an algorithm to calculate the average value of the 10 numbers and to display all the numbers that are greater than the average together with their position in the file. For example, if numbers.txt contained:*

1.3 21.0 7.8 4.5 5.7 8.9 3.5 1.4 4.1 9.0

then the algorithm would display the following:

Average is 6.72
Greater than average & position:
21.0, position 2
7.8, position 3
8.9, position 6
9.0, position 10

Solution:

```
real:value,
     average,
     sum ;
integer:position ;
integer:size IS 10 ;
ARRAY:myNumbers OF [10] real ;
file:numbers ;
numbers ← 'numbers.txt' ;
Open (numbers) ;
sum ← 0 ;
FOR counter GOES FROM 1 TO size
   Get(numbers, REFERENCE:myNumbers [counter - 1] ;
   sum ← sum + myNumbers [counter - 1] ;
ENDFOR
Close (numbers) ;
average ← sum ÷ size ;
Display ('Average is ' + average↲) ;
Display ('Greater than average & position:'↲) ;
FOR counter GOES FROM 1 TO size
   IF (myNumbers [counter - 1] > average
      Display (myNumbers [counter - 1] + ', position ' +
               counter↲) ;
   ENDIF
```

```
    ENDFOR
```

Projects

StockSnackz Vending Machine

Stocksfield Fire Service

Make use of function `Ord (character) RETURNS integer`.

```
ARRAY:fireCodes OF [4] string ← {'Use coarse spray↲',
                                 'Use fine spray↲',
                                 'Use foam↲', 'Use dry agent↲'} ;
Get (keyboard, REFERENCE:EAC) ;
Display (fireCodes[Ordinal(EAC[0])-49]) ;
//Why -1? - think about array indexing

// How do we deal with invalid codes?
IF ((EAC[0] ≥ '1') AND (EAC [0] ≤ '4'))
   Display (fireCodes[Ordinal(EAC[0])-49]) ;
ELSE
   Display ('Invalid fire fighting code↲') ;
ENDIF
```

Or, longhanded:

```
integer:ordinalCode ,
        numericCode ;
character:code ;
code ← EAC [0] ;
ordinalCode ← Ord (code);
numericCode ← ordinalCode -48 // ASCII code 48 = '0', so
                              // subtracting 48 turns character
                              // into numeric equivalent

Display (fireCodes [numericCode -1]) ; // subtract 1 because of
                                       // zero-based array indexing
```

Puzzle World: Roman Numerals & Chronograms

No solutions provided as they are highly dependent on how you have structured your own solutions over the previous chapters.

Pangrams: Holoalphabetic sentences

No solutions provided as they are highly dependent on how you have structured your own solutions over the previous chapters.

Online Bookstore: ISBNs

No solutions provided as they are highly dependent on how you have structured your own solutions over the previous chapters.