Chapter 10

End of Chapter Exercises

1. *What is the difference between a function (expression procedure) and a procedure (command procedure)?*

In short, a procedure goes away and does something; a function goes away and does something and then gives you an answer back. Functions always return a value.

2. *A function is required that takes a single parameter representing a temperature in degrees Fahrenheit and returns the Celsius equivalent. What is missing from the following solution? If the omission is corrected, are there some statements that can be removed?*

```
FUNCTION FahrenheitToCelsius (real fahrenheit)RETURNS real
  real:celsius ;
  celsius ← (fahrenheit - 32) ÷ 1.8 ;
ENDFUNCTION
```

3. *Design a function that takes two integer parameters and returns the larger of the two. What will it do when both arguments have the same value?*

```
FUNCTION Larger (integer:a, integer:b) RETURNS integer
   IF (a > b)
      RETURN a ;
   ELSE
      RETURN b ;
   ENDIF
ENDFUNCTION
```

4. *Write a function that takes four parameters: `day`, `month`, `year` (all integers), and `UKFormat` (a Boolean). The function should combine the three date parameters into a single string which it should then return. The `UKFormat` parameter is used to determined how the string is put together. If `UKFormat` is true then the return string will be in the form dd/mm/yy otherwise it should be in the form mm/dd/yy. For example, if the four arguments were 1, 12, 1984, true (December 1, 1984) then the returned string would be '1/12/1984'. If `UKFormat` were false, then it would return '12/1/1984'.*

```
FUNCTION Date (integer:day, integer:month, integer:year,
                  boolean:UKFormat) RETURNS string ;


   string:theDate ;
   theDate ← '' ;
   IF (UKFormat)
      theDate ← theDate + day + '/' + month + '/' + year ;
   ELSE
```

```
        theDate ← theDate + month + '/' + day + '/' + year ;
    ENDIF
    RETURN theDate ;
ENDFUNCTION
```

5. *Design a function that takes a single string parameter and returns the number of space characters in the string as an integer.*

```
FUNCTION Spaces (string:text) RETURNS integer
    integer:counter,
            numberSpaces ;
    numberSpaces ← 0 ;
    FOR counter GOES FROM 0 TO Length (text) -1
        IF text [counter] = ' '
            numberSpaces ← numberSpaces + 1 ;
        ENDIF
    RETURN numberSpaces ;
ENDFUNCTION
```

6. *A function is required that takes a single parameter representing a temperature in degrees Fahrenheit and returns the Celsius equivalent. What is missing from the following solution? If the omission is corrected, are there some statements that can be removed?*

```
FUNCTION FahrenheitToCelsius (real fahrenheit)RETURNS real
  real:celsius ;
  celsius ← (fahrenheit - 32) ÷ 1.8 ;
ENDFUNCTION
```

The `RETURN` statement is missing:

```
FUNCTION FahrenheitToCelsius (real:fahrenheit)RETURNS real
  real:celsius ;
  celsius ← (fahrenheit - 32) ÷ 1.8 ;
  RETURN celsius ;
ENDFUNCTION
```

We don't need the extra variable Celsius and could write instead:

```
FUNCTION FahrenheitToCelsius (real:fahrenheit)RETURNS real
  RETURN (fahrenheit - 32) ÷ 1.8 ;
ENDFUNCTION
```

7. *Amend Solution 5.9 by adding two procedures `AddSugar` and `AddMilk`. Then replace the statements in the algorithm that deal with adding milk and sugar with calls to the new procedures.*

Replace statements 2.11.1 to 2.11.5 with:

```
2.11.1  AddSugar ;
2.11.2  AddMilk
```

Then add the following two procedures:

```
PROCEDURE AddSugar
   sugarsAdded ← 0 ;
   Get (sugarsRequired) ;
   WHILE (sugarsAdded ≠ sugarsRequired)
      Add 1 spoon sugar ;
      sugarsAdded ← sugarsAdded + 1 ;
   ENDWHILE
ENDPROCEDURE

PROCEDURE AddMilk
   Get (milkRequired) ;
   IF (milkRequired)
      Add milk ;
   ENDIF
ENDPROCEDURE
```

8. *Design a procedure called `swap` that swaps the values of its two integer reference parameters, and then write a line of pseudo-code that invokes the procedure. What would happen if value parameters were used instead of reference parameters?*

Procedure `swap`.

```
PROCEDURE Swap (REFERENCE:integer:a, REFERENCE:integer:b)
   integer:temp ;
   temp ← a ;
   a ← b ;
   b ← temp ;
ENDPROCEDURE
```

If you want to be really perverse and make it hard to understand, try this version which doesn't use a temporary variable:

```
PROCEDURE Swap (REFERENCE:integer:a, REFERENCE:integer:b)
   a ← a + b ;
   b ← a - b ;
   a ← a - b ;
ENDPROCEDURE
```

If you don't believe me, try it out with different values for `a` and `b`. It

doesn't matter if a is greater than or less than b or if a or b is negative

9. *Write a function IsEven that determines whether its single value integer parameter is even or odd. If the parameter is even the function should return a Boolean value True otherwise it should return Boolean False.*

Function IsEven.

```
FUNCTION IsEven (integer:number) RETURNS Boolean ;
IF (number MOD 2 = 0)
   RETURN True ;
ELSE
   RETURN False ;
ENDFUNCTION
```

10. *In the exercises for Chapter 9 you were asked to write an algorithm to convert a lower-case letter into an uppercase one. Assume HTTLAP has two functions Ord and Chr with the following headers:*

FUNCTION Ord (character:aCharacter) RETURNS integer

FUNCTION Chr (integer:ordinalValue) RETURNS character

Ord accepts a single character value in its parameter and returns the ordinal value (ASCII code) of that character. Chr accepts an ASCII value in its integer parameter and returns the corresponding character.

Make use of Ord and Chr to write your own function ToUpper to convert lower-case letters to upper-case. The function should take a single character parameter and return a character which is the upper-case equivalent of the parameter. If the character in the parameter is not a lower-case character then the function should simply give that character back as its return value. For example

myLetter ← ToUpper ('e') ;

would place the character 'E' in myLetter whilst

myLetter ← ToUpper ('Z') ;

would place 'Z' in myLetter and

myLetter ← ToUpper ('3') ;

would place '3' in myLetter.

```
FUNCTION ToUpper (character:letter) RETURNS character
   IF (letter ≥ 'a') AND (letter ≤ 'z')
      RETURN Chr(Ord(letter)-30) ;
   ELSE
```

```
        RETURN letter ;
    ENDIF
ENDFUNCTION
```

11. *Take the van loading solution from Chapter 5 (Solution 5.18) and move the contents of the inner WHILE loop to a sub-program. Decide whether the sub-program should be a procedure or a function and also determine what parameters (if any) it needs.*

Replace statements 5.2.1 to 5.2.3 with:

```
5.2.1  LoadVan (REFERENCE:payload, parcelWeight);
5.2.2  Get next parcelWeight ;
```

Then add this procedure:

```
PROCEDURE LoadVan (REFERENCE:integer:load, integer:weight)
   Load parcel on van ;
   load ← load + weight ;
ENDPROCEDURE
```

Projects

StockSnackz Vending Machine

Amend your solution to take into account any necessary data types introduced in this chapter. Consider carefully the data types needed to handle the monetary values.

```
integer:chocolateStock,
        muesliStock,
        cheesePuffStock,
        appleStock,
        popcornStock ;
```

Stocksfield Fire Service

Write your solution to the EAC decoding problem as an algorithm using the more formalized HTTLAP pseudo-code introduced in this chapter. Make sure you declare all your variables with appropriate data types. You need to think carefully about what you are going to use to store the EAC. The easiest method to get you started is to store each of the three characters of the EAC in separate character variables.

```
character:fireFightingCode,
          precautionsCode,
          publicHazardCode ;
```

```
Get (keyboard, REFERENCE:fireFightingCode,
REFERENCE:precautionsCode, REFERENCE:publicHazardCode) ;
// First character
IF (fireFightingCode = '1')
   Display ('Use coarse spray↲') ;
ELSE IF (fireFightingCode = '2' )
   Display ('Use fine spray↲') ;
ELSE IF (fireFightingCode = '3')
   Display ('Use foam↲') ;
ELSE IF (fireFightingCode = '4')
   Display ('Use dry agent↲') ;
ELSE
   Display ('Invalid fire fighting code↲') ;
ENDIF
// Second character
IF (precautionsCode = 'P')
   Display ('Use LTS↲') ;
   Display (Dilute 'spillage↲') ;
   Display ('Risk of explosion↲') ;
ELSE IF…
…
ELSE IF (precautionsCode = 'Z')
   Display ('Use BA & Fire kit↲') ;
   Display ('Contain spillage↲') ;
ELSE
   Display ('Invalid precautions code↲') ;
ENDIF
// Third character
IF (publicHazardCode = 'E')
   Display ('Public hazard↲') ;
ELSE IF (publicHazardCode = ' ')
   Display ('No hazard↲') ;
ELSE
   Display ('Invalid public hazard code↲') ;
ENDIF
```

Alternatively, we could access the EAC as a string:

```
string:EAC ;
Get (keyboard, REFERENCE:EAC) ;
IF (EAC[0]= '1')
   Display…
```

Puzzle World: Roman numerals & chronograms

Chronograms (also called eteostichons) are sentences in which certain letters, when rearranged, stand for a date and the sentence itself is about the subject to which the date refers. All letters that are also roman numerals (I, V, X, L, C, D, M) are used to form the date. Sometimes the sentence is written such that the roman numeral letters already give a well-formed roman number. For example, in the sentence:

My Day Closed Is In Immortality

if we ignore the lower-case letters we get the number MDCIII which equals 1603. The sentence commemorates the death of Queen Elizabeth the First of England in 1603. More commonly, the roman numbers are not well formed and the date is obtained by adding the values of all the roman numerals in the sentence, as in:

LorD haVe MercI Vpon Vs. (V used as a U, mercy spelt with an 'i')

This is a chronogram about the Great Fire of London in 1666. The date is given by L+D+V+M+I+V+V = 50 + 500 + 5 + 1000 + 1 + 5 + 5 = 1666.

Outline the basic algorithm for finding and displaying in decimal the date 'hidden' in a chronogram. To begin, assume that only upper-case letters are used for roman numerals (I=1, but i is a letter). Also, assume that the roman numerals do not have to form a valid string of numerals and that the hidden date is obtained simply by summing the values of all roman numerals found (as in the 'Lord have mercy upon us' example above).

For an extra challenge, extend this solution to accept only chronograms that have a well-formed roman number in them. Thus "My Day Closed Is In Immortality" would give the valid date MDCIII, whilst "LorD haVe MercI Vpon Vs" would not give us a result as LDVMIVV is not a well-formed number (1666 should be written as MDCLXVI).

Basic problem:

```
string:chronogram ;
integer:counter,
        value ;
char:current ;
value ← 0 ;

chronogram ← 'LorD haVe MercI Vpon Vs' ;
FOR counter GOES FROM 0 TO Length (chronogram) - 1
   current ← chronogram [counter] ;
   IF (current ≥'A') AND (current ≤ 'Z')
```

```
      value = value + value of current digit ;
   ENDIF
ENDFOR
```

Extended version:

```
string:chronogram,
       numberString ;
integer:counter,
        value ;
char:current ;
value ← 0 ;
numberString ← '' ;

chronogram ← 'LorD haVe MercI Vpon Vs' ;
FOR counter GOES FROM 0 TO Length (chronogram) - 1
   current ← chronogram [counter] ;
   IF (current ≥'A') AND (current ≤ 'Z')
      numberString ← numberString + current ;
   ENDIF
ENDFOR
```

Now we have the hidden date stored in `numberString` we can simply validate and decode it as per our previous algorithms.

Pangrams: holoalphabetic sentences

Online bookstore: ISBNs