Chapter 9

End of Chapter Exercises

1. *What is the difference between a literal and a variable?*

A variable is an area of memory set aside to hold values. The value of the variable can be changed as the program runs. The value held in the variable is accessed via the variable's name, or identifier. A literal is simply a raw value which is not stored anywhere. For example, in the statement

```
number ← 3 + number2 ;
```

the variable `number` is assigned the value of the expression `3 + number2` where `number2` is also a variable and 3 is a literal value.

2. *Could values such as 5.5, 3.142, 1090.1 be assigned to the following variable?*

```
integer:intNumber ;
```

No. They are real values and `intNumber` is an integer.

3. *Why are real numbers not ordinal?*

Because the ordinal numbers are 'counting' numbers and denote relative position in a sequence. The integers are ordinal because we know that the number 7 comes after the number 6. Real numbers, on the other hand, cannot be counted in this way as between 6.0 and 7.0 there is an infinite set of real values (e.g. 6.000000000000000000001, 6.200000000006, and so on) so it is not possible to count them in the same way that ordinal numbers can be counted.

4. *What are the largest positive values that can be stored in the following variables?*

```
integer:number ;

shortint:number2 ;

byte:number3 ;

longint:number4 ;
```

`integer` = 2,147,483,647, `shortint` = 32,767, `byte` = 127, `longint` = 9,223,372,036,854,775,807. See Table 9.2.

5. *What is the difference between the `integer` value 4 and the `character` '4'?*

Integer 4 is, well, the number 4. Character 4 is a symbol used to display the number 4 on the screen, on a printer, etc. Integer variables are used to store numeric values whilst character variables store character symbols. The characters, as well as being the digits 0 to 9 also comprise letters, punctuation symbols, and so forth. Groups of characters form strings, whilst groups of numbers form larger numbers.

6. *What is the `character` type used for?*

Storing individual occurrences of character symbols. For instance, you might want to keep a loop going every time the user presses the 'Y' key on the keyboard in response to a prompt. The key pressed by the user could be stored in a character variable.

7. *Declare a `string` variable `myName` and assign your name to it.*

You could do this 2 ways in the *HTTLAP* pseudo-code:

```
string:myName ;
myName ← 'Paul' ;
```

or

```
string:myName ← 'Paul' ;
```

8. *Given the following declaration and assignment statements:*

```
boolean:onVacation ;
onVaction ← true ;
```

How can the following if statement be improved?

```
IF (onVacation = true)
  Display ('No milk today, thankyou') ;
ENDIF
```

Simple, just remove the '= true' part:

```
IF (onVacation)
     Display ('No milk today, thankyou') ;
ENDIF
```

9. *Given the declaration:*

```
integer:membersOfFamily IS 5 ;
```

What is wrong with the following statement?

```
membersOfFamily ← membersOfFamily + 1 ;
```

`membersOfFamily` is a constant not a variable (double border is a clue). Constant values are immutable (they cannot be changed) and so remain constant throughout (hence their name). Therefore, once defined, a constant identifier cannot have its value changed.

10. *Given the declaration:*

integer: $\boxed{\texttt{result}}$;

For each of the following statements state what value will be assigned to `result`.

1. $\boxed{\texttt{result}} \leftarrow 3 + 4 \times 7$;

2. $\boxed{\texttt{result}} \leftarrow 4 \times 7 + 3$;

3. $\boxed{\texttt{result}} \leftarrow (4 \times 7) + 3$;

4. $\boxed{\texttt{result}} \leftarrow 4 \times (7 + 3)$;

5. $\boxed{\texttt{result}} \leftarrow 4 \times 7 + 3 \times 8$;

6. $\boxed{\texttt{result}} \leftarrow 4 \times (7 + 3) \times 8$;

7. $\boxed{\texttt{result}} \leftarrow 4 \times 7 \div 2 \times 3 \div 21$;

8. $\boxed{\texttt{result}} \leftarrow 4 \times 7 \div 2 \times 3\wedge3 \div 21$;

1: Multiplication comes before addition, so 3 + 28 = 31
2: Multiplication comes before addition, so 28 + 3 = 31
3: Expressions in parentheses evaluated first, so 28 + 3 = 31
4: Expressions in parentheses evaluated first , so 4 times 10 = 40
5: Multiplications done before the addition, so 28 + 24 = 52
6: Parentheses, then multiplication, then addition, so 4 times 10 times 8 = 320
7: All operators here of equal precedence so just work from left to right, so 28 divided by 2 = 14, then multiply by 3 = 42, then divide by 21 = 2
8: exponentiation comes first, so 3^3 = 27. Then work left to right: 28 divided by 2 = 14, multiply by 27 = 378, then divide by 21 = 18

11. *Given the declaration:*

boolean: $\boxed{\texttt{bigger}}$;

For each of the following statements state what value will be assigned to `bigger`.

1. $\boxed{\texttt{bigger}} \leftarrow 3 > 4$;

```
2. bigger ← 3 × 2 > 4 ;

3. bigger ← 'a' > 'Z'  ;

4. bigger ← 3 × 4 = 2 × 6 ;
```

1: `bigger` will be assigned the value 'False' because 3 is not greater than 4
2: 6 is greater than 4, so True
3: True. Character 'a' has position (ordinal value) 97 in the ASCII character set while 'Z' is character 90 (see Table 9.5). 97 is greater than 90, so the test evaluates to True.
3: 3 times 4 = 12 and 2 times 6 = 12. 12 does equal 12, so the value True is assigned to `bigger`.

12. *Anyone resident in the UK and who is a citizen of the UK, the Republic of Ireland or of a Commonwealth country and is aged 18 or over on the date of a parliamentary election is eligible to vote, unless they are a member of the House of Lords, imprisoned for a criminal offence, mentally incapable of making a reasoned judgement, or have been convicted of corrupt or illegal practices in connection with an election within the previous five years. Declare variables to represent the components of this problem. Then write an assignment statement that uses these variables in a relational expression to assign the value True or False as appropriate to the Boolean variable `eligibleToVote` which will store an individual's eligibility to vote. Finally, draw a table showing different values of the variables and the Boolean value that would be assigned to `eligibleToVote` as a result.*

The principal components of the problem are to do with whether the person is an adult, is a resident, is a Lord, is imprisoned, has a conviction, and is mentally incapable. We can declare Boolean variables to hold these values as well as the variable `eligibleToVote`:

```
boolean:isAdult,
        isResident,
        isALord,
        isImprisoned,
        isMentallyIncapable,
        hasConviction,
        eligibleToVote ;
```

Assuming the program has assigned values to the first six variables, we can assign `eligibleToVote` thus:

```
eligibleToVote ← ((isResident) AND (isAdult)
                                  AND NOT (isALord)

                   AND NOT (isImprisoned) AND

                   NOT (isMentallyIncapable) AND

                   NOT (hasConviction)) ;
```

With all the `NOT`s we could use parentheses and DeMorgan to simplify the expression:

```
eligibleToVote ← ((isResident) AND (isAdult) AND

                   NOT ((isALord) OR (isImprisoned) OR

                   (isMentallyIncapable) OR

                   (hasConviction)) ;
```

13. *Think about this problem of representation. Why do you think it is that real numbers are often only stored as an approximation of their true value? Can you think of ways that exact real numbers could be represented? For example, the fraction 1/3 leads to an approximation when represented in the decimal number system (0.333333333…), whilst the fraction 1/10 can be represented exactly in decimal (0.1) but not in binary.*

In decimal unless the real number can be expressed as an exact power of 10 we end up with these recurring sequences, e.g. 1/3 = 0.33333r. The same goes for binary. If the real number is exact and can be expressed as an exact power of 2 then it will work, if not then we get approximations again.

14. *Design an algorithm that will convert any lower-case letter to its upper-case equivalent. Use a character variable `theLetter` to hold the letter. For example, if `theLetter` was assigned the value `'e'` the algorithm would assign it a new value of `'E'`. Before converting the lower-case letter, first ensure it really is a lower-case letter and not some other character.*

The first thing I would do it using a real programming language would be to look through the language's manual and see if it already has existing functions or procedures for doing this. Most languages have a `ToUpper` routine of some description. If this were the case then it would be a simple matter of:

```
theLetter ← ToUpper (theLetter) ;
```

job done! However, to do it from scratch we need to use what we know about character codes. All the lower case letters have ASCII codes between 97 ('a') and 122 ('z'). The upper case letters live in the range 67 ('A') to 90 ('Z'). The simple form of the problem is to assume that `theLetter` contains a lower case letter we just need to set it to the character 30 places earlier in the character set. For example, ‘a’ = 97, subtract 30 gives 67 which is ‘A’. The full problem requires us first to ascertain whether the character in theLetter is really a lower case letter. Put this all together using the Ord and Chr routines discussed in section 9.1.5and and we get:

```
integer:newCharNumber ;

IF (theLetter ≥ 'a') AND (theLetter ≤ 'z')

   newCharNumber ← Ord (theLetter) – 30 ;

   theLetter ← Chr (newCharNumber) ;

ENDIF
```

We could do it without the new variable `newCharNumber` like this:

```
IF (theLetter ≥ 'a') AND (theLetter ≤ 'z')

   theLetter ← Chr (Ord (theLetter) – 30) ;

ENDIF
```

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

Update the variable list for your pangram solution by assigning proper HTTLAP types. Rewrite your algorithm using the more formal HTTLAP pseudo-code, assignment statements, Boolean expressions in `IF` and `WHILE` conditions, and so forth.

```
string:sentence ;
character:current,
          userResponse ;
integer:counter,
        notCrossedout ;
DO
    Get (keyboard, REFERENCE:sentence) ;
    FOR current GOES FROM 'A' TO 'Z' ;
        Write current
    ENDFOR
```

```
   FOR counter GOES FROM 0 TO Length (sentence) -1
      current ← sentence [counter] ;
      Cross off letter on the paper that matches current letter ;
   ENDFOR
   notCrossedOut ← 0 ;
   FOR current GOES FROM 'A' TO 'Z' ;
       IF (current not crossed out)
          notCrossedOut ← notCrossedOut + 1 ;
       ENDIF
   ENDFOR
   IF (notCrossedOut = 0)
       Display ('Sentence is a pangram') ;
   ELSE
       Display ('Sentence is NOT a pangram') ;
   ENDIF
   Display ('Do you want to test another sentence? Y/N') ;
   Get (keyboard, REFERENCE:userResponse) ;
WHILE (userResponse ≠ 'N') ;
```

Online bookstore: ISBNs

Validation problem

```
string:ISBN ;
integer:total,
        counter,
        remainder ,
        calcCheck ;
character:currentDigit,
          checkDigit,
          calcCheckDigit ;

Display ('Enter a ten-character ISBN') ;
Get (keyboard, REFERENCE:ISBN) ;
total ← 0 ;
FOR counter GOES FROM 1 TO 9
   currentDigit ← ISBN [counter] ;
   total ← total + Ord(currentDigit) - 48 ;
ENDFOR
checkDigit ← ISBN [9] ;
remainder ← 11 – (total ÷ 11) ;
```

```
calcCheck ← 11 – remainder ;
IF (calcCheck = 10)
   calcCheckDigit ← 'X' ;
ELSE
   calcCheckDigit ← Chr (calcCheck + 48) ;
ENDIF
IF (calcCheck = checkDigit)
   Display ('ISBN valid') ;
ENDIF
```