# Chapter 6

## End of Chapter Exercises

**1.** *There are two things wrong with the following* IF...ELSE *statement. What are they?*

```
IF (mark ≥ 40)
   Display 'You have passed.' ;
ELSE IF (mark ≤ 40)
   Display 'You have failed.' ;
ENDIF
```

First it is ambiguous – what should happen when `mark` is 40 as 40 satisfies both conditions? Second, the `ELSE` doesn't need the `IF`: it should just be

```
IF (mark ≥ 40)
   Display 'You have passed.' ;
ELSE
   Display 'You have failed.' ;
ENDIF
```

**2.** *When should you use a FOR loop rather than a WHILE or DO…WHILE loop? When should you used a DO…WHILE rather than a WHILE or FOR loop?*

FOR loop: when you want to implement a count-controlled loop and the number of repetitions is known in advance or can be determined prior to the start of the loop.

`DO...WHILE`: When you want to iteratate an undetermined number of times but at least once.

`DO...WHILE` is a 1-or-more loop, while the `WHILE` is a zero-or-more loop.

**3.** *As any loop written using a* FOR *or a* DO...WHILE *can also be constructed using the basic* WHILE *construct, what is the benefit of using the* FOR *and* DO...WHILE*?*

They allow a more precise control abstraction. `WHILE` offers a general control abstraction that can be used to implement any iteration, but if upon examination the nature of the loop reveals certain characteristics (such as those listed in question 2) then use the appropriate iteration construct. The principle here is use the control abstraction that most closely fits the problem.

**4.** *State the difference between a determinate and an indeterminate loop.*

A determinate loop is one in which the number of iterations is known or can be calculated before the loop executes. An indeterminate loop is one in which the number of iterations is not known or cannot be calculated in advance. For example, a loop to calculate the average temperature for each day of the year would be determinate as we know it needs to run 365 or 366 times depending on whether it is a leap year or not. Even though it could be either 365 or 366 it is still determinate because before the loop begins its first execution the number of iterations has been calculated. An indeterminate loop could be one in a program that asks people how they will vote in the next election, stopping only when a certain time of day is reached or no-one has expressed a preference for, say, 1 hour.

**5.** *In Stocksfield High School all the children's work is marked out of 5. An A grade is awarded for work scoring between 4 and 5. A score of at least 3 would get a B grade, and a C is given for a score of at least 2. Any other score gets an F (fail). For example, scores of 4.0, 4.8, and 5 would get grade A, 3.2 and 3.9 would get a B, 2.9 a C and 1.9 an F. Write an* `IF…ELSE` *construct that assigns the appropriate grade to the variable* `studentGrade` *depending on the value of the variable* `studentMark`*.*

```
IF (studentMark ≥ 4) // A Grade
   studentGrade ← A ;
ELSE IF (studentMark ≥ 3) // B Grade
   studentGrade ← B ;
ELSE IF (studentMark ≥ 2) // C Grade
   studentGrade ← C ;
ELSE //Anything under 2 is a fail
   studentMark ← F ;
ENDIF
```

**6.** *Stocksfield High School has just changed its grading system. Now, an A grade is given for marks between 4 and 5, a B for marks of at least 3.5, a C for marks of at least 3.0, a new pass grade D for marks of at least 2.0 and an F for every mark lower than 2.0 Amend your solution to exercise 5 to account for this change.*

```
IF (studentMark ≥ 4) // A Grade
   studentGrade  A ;
ELSE IF (studentMark ≥ 3.5) // B Grade
   studentGrade ← B ;
ELSE IF (studentMark ≥ 3) // C Grade
   studentGrade ← C ;
ELSE IF (studentMark ≥ 2) // D Grade
   studentGrade ← D ;
ELSE //Anything under 2 is a fail
   studentMark ← F ;
ENDIF
```

**7.** *Write your solution to exercise 6 using a sequence of* `IF` *statements without* `ELSE` *parts. What do you have to do to the condition of each* `IF` *statement in order to ensure they work properly?*

```
IF (studentMark ≥ 4) // A Grade
   studentGrade ← A ;
ENDIF
IF (studentMark ≥ 3.5) AND (studentMark < 4) // B Grade
   studentGrade ← B ;
ENDIF
IF (studentMark ≥ 3) AND (studentMark < 3.5) // C Grade
   studentGrade ← C ;
ENDIF
IF (studentMark ≥ 2) AND (studentMark < 3) // D Grade
   studentGrade ← D ;
ENDIF
IF (student < 2) //Anything under 2 is a fail
   studentMark ← F ;
ENDIF
```

The conditions have to include both the upper and lower limits for each grade.

**8.** *Extend your solution to exercise 5 to assign to calculate and display the student grades for a class of twenty students.*

As we know it is for 20 students, we could use the specialised determinate iteration abstraction -- the FOR loop:

```
numberGrades ← 0 ;
classSize ← 20 ;
FOR numberGrades GOES FROM 1 TO classSize
   Get studentMark ;
   IF (studentMark ≥ 4) // A Grade
      studentGrade ← A ;
   ELSE IF (studentMark ≥ 3.5) // B Grade
      studentGrade ← B ;
   ELSE IF (studentMark ≥ 3) // C Grade
      studentGrade ← C ;
   ELSE IF (studentMark ≥ 2) // D Grade
      studentGrade ← D ;
   ELSE //Anything under 2 is a fail
      studentMark ← F ;
   ENDIF
ENDFOR
```

**9.** *Using HTTLAP design an algorithm for a guessing game. The algorithm should choose a number between 1 and 10 and the player has three tries in which to guess the number. If the player guesses correctly then the response 'Correct!' should be given. When an incorrect guess is made the response should be 'Hot!' if the guess was 1 away from the actual number, 'Warm…' when the guess is two away, and 'COLD!' when the guess is 3 or more away. Make sure your loop finishes if the player guesses correctly in less than three turns.*

The basic algorithm might look like this:

```
maxGuesses ← 3 ;
numberGuesses ← 0 ;
userGuess ← 0 ;
number ← random number between 1 and 10 ;
correct ← 0 ; // 0 = wrong, 1 = correct guess.
DO
   Display 'Enter your guess ' ;
   Get userGuess ;
   IF (userGuess 3 or more away from number)
      Display 'COLD!' ;
   ELSE IF (userGuess within 2 of number)
      Display 'Warm...' ;
   ELSE IF (userGuess within 1 of number)
      Display 'Hot!' ;
   ELSE
      Display 'Correct!' ;
      correct ← 1 ;
   ENDIF
WHILE (numberGuesses ≤ maxGuesses) AND (correct = 0) ;
IF (correct = 0)
   Display 'You failed.' ;
ENDIF
```

I used a DO...WHILE loop because the user needs at least 1 guess. The remaining problem is how to express the 'within 3 of number' type of condition. How do we know if the guess is 3 or more away? Say the computer's number were 7 and the user guessed 4. Subtracting 4 from 7 gives 3 which means we need a 'COLD!' response. But what if the computer picked 2 and the user picked 4? Subtracting 4 from 2 gives -2 which is 4 away from 2, yet 4 is only 2 away which deserves a 'Warm...' response. We only want to subtract the user's guess from the computer's number when it is smaller otherwise it needs to be the other way round. What we need is to find the **absolute** difference. Many programming languages have an 'abs' function that takes strips the sign off a number and gives it back. So, the absolute value of 2-4 is 2 because 2-4 =-2 and stripping the sign off gives 2. We could assume our pseudo-code has such a facility, or we could be real programmers and work out how to do it. To find the absolute difference between two numbers all we have to do is subtract the smaller one from the larger one. We could do it this way:

```
IF (userGuess ≥ number)
   difference ← userGuess – number ;
ELSE
   difference ← number – userGuess ;
ENDIF
```

We could then use this solution in our algorithm above like this:

```
difference ← 0 ;
maxGuesses ← 3 ;
numberGuesses ← 0 ;
userGuess ← 0 ;
number ← random number between 1 and 10 ;
correct ← 0 ; // 0 = wrong, 1 = correct guess.
DO
   Display 'Enter your guess ' ;
   Get userGuess ;
   IF (userGuess ≥ number)
      difference ← userGuess – number ;
   ELSE
      difference ← number – userGuess ;
   ENDIF
   IF (difference ≥ 3)
      Display 'COLD!' ;
   ELSE IF (difference = 2)
      Display 'Warm...' ;
   ELSE IF (difference = 1)
      Display 'Hot!' ;
   ELSE
      Display 'Correct!' ;
      correct ← 1 ;
   ENDIF
   numberGuesses ← numberGuesses + 1 ;
WHILE (numberGuesses < maxGuesses) AND (correct = 0) ;
IF (correct = 0)
   Display 'You failed.' ;
ENDIF
```

**10.** Honest Brian's Insurance *sells car insurance policies. The company is owned by Brian who is a cautious type and so hikes the premiums for young male drivers (who are statistically much more likely to make a claim). The basic premium charge is 3% of the value of the vehicle. This figure is raised by 11% for male drivers under 25 years of age, and by 6% for female drivers under the age of 21. A further £/$/€ 250 is added to the premium of any driver who has had any kind of speeding ticket. Design an algorithm to calculate the insurance premiums for the drivers in Table 6.5 (which also shows the premium your algorithm should generate).*

| **Name** | **Age** | **Sex** | **Car Value** | **Speeding ticket?** | **Premium** |
|---|---|---|---|---|---|
| Nick | 60 | M | 15,000 | No | £450.00 |
| Chris24 | M | 24,000 | No | £799.20 | |
| Lynne | 23 | F | 8,000 | No | £240.00 |
| Alf | 17 | M | 35,000 | Yes | £1,415.50 |
| Shadi | 23 | M | 15,000 | No | £499.50 |
| Becky | 18 | F | 12,000 | Yes | £631.60 |

```
premium ← carValue × 0.03 ;
IF (driverSex = 'M') AND (driverAge < 25)
   premium ← premium × 1.11 ;
ELSE IF (driverSex = 'F') AND (driverAge < 21)
   premium  ← premium × 1.06 ;
ENDIF
IF (speedingTicket = 'Yes')
   premium  ← premium + 250 ;
ENDIF
```

**11.** *The Beaufort scale is used to classify wind speeds. Use Table 6.6 to design an algorithm that declares a variable windSpeed, assigns that variable a value representing the wind speed in miles-per-hour (use zero for values < 1) and then displays the corresponding Beaufort number and description. For example, if windSpeed had the value 20 the algorithm would display*

*Beaufort scale:5, Fresh breeze.*

*Hint: the conditions in your selections will need to be compound.*

| ***Beaufort scale*** | ***Wind speed (miles per hour)*** | ***Description*** |
|---|---|---|
| *0* | *<1* | *Calm* |
| *1* | *1-3* | *Light air* |
| *2* | *4-7* | *Light breeze* |
| *3* | *8-12* | *Gentle breeze* |

| | | |
|---|---|---|
| *4* | *13-18* | *Moderate breeze* |
| *5* | *19-24* | *Fresh breeze* |
| *6* | *25-31* | *Strong breeze* |
| *7* | *32-38* | *Near gale* |
| *8* | *39-46* | *Gale* |
| *9* | *47-64* | *Strong gale* |
| *10* | *55-63* | *Storm* |
| *11* | *64-72* | *Violent storm* |
| *12* | *≥73* | *Hurricane* |

```
Display 'Enter a wind speed' ;
windSpeed ← value typed by user ;
IF ( windSpeed = 0)
   Display 'Beaufort scale:0, Calm' ;
ELSE IF (windSpeed ≤ 3)
   Display 'Beaufort scale:1, Light air'  ;
ELSE IF (windSpeed ≤ 7)
   Display 'Beaufort scale:2, Light breeze' ;
ELSE IF (windSpeed ≤ 12)
   Display 'Beaufort scale:3, Gentle breeze' ;
ELSE IF (windSpeed ≤ 18)
   Display 'Beaufort scale:4, Moderate breeze' ;
ELSE IF (windSpeed ≤ 24)
   Display 'Beaufort scale:5, Fresh breeze' ;
ELSE IF (windSpeed ≤ 31)
   Display 'Beaufort scale:6, Strong breeze' ;
ELSE IF (windSpeed ≤ 39)
   Display 'Beaufort scale:7, Near gale' ;
ELSE IF (windSpeed ≤ 46)
   Display 'Beaufort scale:8, Gale' ;
ELSE IF (windSpeed ≤ 54)
   Display 'Beaufort scale:9, Strong Gales' ;
ELSE IF (windSpeed ≤ 63)
   Display 'Beaufort scale:10, Storm' ;
ELSE IF (windSpeed ≤ 72)
   Display 'Beaufort scale:11, Violent storm' ;
ELSE
      Display 'Beaufort scale:12, Hurricane' ;
ENDIF
```

**12.** *You have been asked by an online bookstore to design an algorithm that calculates postage costs. Postage is calculated by adding a fixed weight-based charge to a* *handling fee for each book in the order. Table 6.7 shows how the charge is calculated.*

| ***Weight*** | ***Postage rate*** | ***Handling fee per book*** |
|---|---|---|
| *Up to 200g* | *1.75* | *0.25* |
| *Up to 400g* | *2.50* | *0.40* |
| *Up to 1000g* | *4.00* | *0.45* |
| *Over 1000g* | *6.00* | *0.5* |

*For example, an order for one book weighing 180 g would cost 1.75 + 0.25 = 2.00 to deliver. An order weighing 1500 g total with three books in it would cost 6.00 + 0.5 × 3 = 7.50 to deliver. Using* HTTLAP *design an algorithm that calculates the postage charge for book orders. The parcel weight and number of books in the order should be provided by the user. The algorithm should calculate postage costs for an unspecified number of orders, terminating when the parcel weight given by the user is a negative number.*

```
Get numberBooks  ;
Get weight ;
WHILE (weight > 0)
  IF (weight ≤ 200)
      postageRate ← 1.75 ;
      handlingFee ← 0.25 × numberBooks ;
  ELSE IF (weight ≤ 400)
      postageRate ← 2.5 ;
      handlingFee ← 0.4 × numberBooks ;
  ELSE IF (weight ≤ 1000)
      postageRate ←  4.0 ;
      handlingFee ← 0.45 × numberBooks ;
  ELSE
      postageRate ←  6.0
      handlingFee ← 0.5 × numberBooks ;
  ENDIF
  postageCharge ← 0.25 postageRate + handlingFee ;
  Get numberBooks  ;
  Get weight ;
ENDWHILE
```

**13.** *Stocksfield High School gives email addresses to all its pupils in the form:*

*givenName.familyName@stocksfieldhigh.ac.uk*

*Teachers are given email addresses of the form:*

*firstInitial.secondInitial.familyName@stocksfieldhigh.ac.uk*

*where the second initial may not be present.*

*Table 6.8 shows some example teacher and pupil names and the corresponding email addresses.*

| ***Teachers*** | ***email*** | ***Pupils*** | ***email*** |
|---|---|---|---|
| *Henry Higgins* | *h.higgins@...* | *Emily Harris* | *emily.harris@...* |
| *Alfred P. Doolittle* | *a.p.doolittle@...* | *Sarah Jane Smith* | *sarah.smith@...* |
| *Jennifer P.D. Quick* | *j.p.quick@...* | *John Dobby* | *John.dobby@...* |

*Design an algorithm that declares a variable emailAddress, assigns to the variable a value provided by the user, and displays the family name of the owner of that email address and whether they are a pupil or a teacher. For example, if the user typed in*

```
emily.harris@stocksfieldhigh.ac.uk
```

*the algorithm would display*

```
Harris:Pupil.
```

*If the user typed*

```
a.p.doolittle@stocksfieldhigh.ac.uk
```

*The algorithm would display*

```
Doolittle:Teacher.
```

*Note, the algorithm should be able to handle any valid Stocksfield High email address, not just the ones listed in Table 6.8.*

This is quite a tough algorithm if you try and specify it completely. Really, we need some knowledge of how to process strings of characters which is not covered in this book (though it is in Part II of the longer version of this book titled "How to think like a programmer: program design solutions for the bewildered"). The best way to tackle it is at a high level, as if you were doing it yourself. The difference between staff and pupils is that pupil email addresses show their complete first whereas staff email addresses just show the initials of their first names. Here's a way forward:

```
Get emailAddress ;
Display characters between '@' and preceding '.' ;  // that's
the family name
IF (length of name before first '.' > 1 character) // dealing
with a pupil
   Display ':Pupil.' ;
ELSE // dealing with staff
   Display ':Teacher.' ;
ENDIF
```

That's all we need for the general algorithm. If you want more of a challenge, then consider how you would go about splitting the email address up into its component parts.

**14.** *Design an algorithm that declares an integer variable* `timesTable`*, gets a value for this variable, and displays the times table up to 15 × n. For example, if* `timesTable` *had the value 12, then the following output would be displayed:*

```
1 × 12 = 12
2 × 12 = 24
3 × 12 = 36
4 × 12 = 48
5 × 12 = 60
6 × 12 = 72
7 × 12 = 84
8 × 12 = 96
9 × 12 = 108
10 × 12 = 120
11 × 12 = 132
12 × 12 = 144
13 × 12 = 156
14 × 12 = 168
15 × 12 = 180
```

```
Get timesTable ;
FOR counter GOES FROM 1 TO timesTable
   product ← counter × 12 ;
   Display counter ' × ' 12 ' = ' product ;
ENDFOR
```

## Projects

### StockSnackz Vending Machine

*Now that you have been introduced to the* `IF...ELSE` *construct, revise your previous vending machine solution to deal more elegantly with the problem of dispensing the chosen snack. Using* `IF...ELSE` *will also make the problem of dealing with buttons 0, 7, 8, and 9 much simpler.*

*Now that a proper selection construct has been used it is time to make the vending machine much more interesting. The University of Stocksfield is losing too much money through greedy staff stocking up on free snacks. With the exception of the sales summary (button 6) all items must now be paid for and cost 10 pence each. If a user presses a button for a snack before sufficient money has been inserted an 'insufficient funds' message should be displayed. If the user has deposited sufficient money for an item then the machine will dispense the chosen snack. Assume no change is given. Extend your solution to reflect these new requirements.*

*Now extend your solution so that if the user has deposited more than 10 pence/cents the machine gives any required change after dispensing the chosen item. You may assume that the machine always has sufficient stock of each denomination of coin to be able to make exact change. The machine accepts (and gives back) the following denominations of coins: 1, 2, 5, 10, 20, 50 pence*
*. Change should be dispensed using the fewest coins possible. Note, you have already solved the change-giving problem (see the exercises for Chapter 3) so see if that solution can be reused (perhaps with some amendments) here. There is a mathematical operator called modulo which gives the remainder after division. It often has the symbol %, but our pseudo-code used MOD. It works like this: 20 ÷ 7 = 2 : 7 goes into 20 twice. 20 MOD 7 = 6 : the left over after dividing 20 by 7 is 6. You will find this useful for working out how to give change.*

*In the United Kingdom and countries using the euro currency another two demonations of coin are available. The UK has £1 and £2 coins and the Euro Zone similarly has €1 and €2 coins. Extend your solution to cater for these larger denomination coins. If your machine works on US dollars and you would like to accept the rarer half-dollar and one-dollar coins, by all means go ahead.*

**Use IF…ELSE:**

```
1. Install machine ;
2. Turn on power ;
3. Fill machine ;
4. chocolateStock ← 5 ;
5. muesliStock ← 5 ;
6. cheesePuffStock ← 5 ;
```

```
7. appleStock ← 5 ;
8. popcornStock ← 5 ;
9. WHILE (not the end of the day)
   9.1 IF (button 1 pressed)
          IF (chocolateStock > 0)
             Dispense milk chocolate ;
             chocolateStock ← chocolateStock - 1 ;
          ELSE
             Display 'Sold out message' ;
          ENDIF
   9.2 ELSE IF (button 2 pressed)
          IF (muesliStock > 0)
             Dispense muesli bar ;
             muesliStock ← muesliStock - 1 ;
          ELSE
             Display 'Sold out message' ;
          ENDIF
   9.3 ELSE IF (button 3 pressed)
          IF (cheesePuffStock > 0)
             Dispense cheese puffs ;
             cheesePuffStock ← cheesePuffStock - 1 ;
          ELSE
             Display 'Sold out message' ;
          ENDIF
   9.4 ELSE IF (button 4 pressed)
          IF (appleStock > 0)
             Dispense apple ;
             appleStock ← appleStock - 1 ;
          ELSE
             Display 'Sold out message' ;
          ENDIF
   9.5 ELSE IF (button 5 pressed)
          IF (popcornStock >0)
             Dispense popcorn ;
             popcornStock ← popcornStock - 1 ;
          ELSE
             Display 'Sold out message' ;
          ENDIF
   9.6 ELSE IF (button 6 pressed)
          Print sales summary ;
   9.7 ELSE // we know an invalid button was pushed, no need to
   test for it.
          Display 'Invalid choice message' ;
```

```
      ENDIF
   ENDWHILE
```

**Checking for sufficient money (only partial solution given for effect):**

```
9. priceOfChocolateBar ← 10 ;
10. WHILE (not the end of the day)
    10.1 IF (button 1 pressed)
           IF (money > priceOfChocolateBar)
              IF (chocolateStock > 0)
                 Dispense milk chocolate ;
                 chocolateStock ← chocolateStock - 1 ;
              ELSE
                 Display 'Sold out message' ;
              ENDIF
           ELSE
              Display 'Insufficient funds' message ;
           ENDIF
    10.2 ELSE IF (button 2 pressed)
      . . .
      . . .
      . . .
```

**Algorithm for giving change:**

```
1. leftover  money - priceOfSnack ;
2. numberFifties ← leftover ÷ 50 ;
3. leftover ← change MOD 50 ;
4. numberTwenties ← leftover ÷ 20 ;
5. leftover ← leftover MOD 20 ;
6. numberTens ← leftover ÷ 10 ;
7. leftover ← leftover MOD 10 ;
8. numberFives ← leftover ÷ 5 ;
9. leftover  leftover MOD 5 ;
10. numberTwos ← leftover ÷ 2 ;
11. numberOnes ← leftover MOD 2 ;
12. FOR count GOES FROM 1 TO numberFifties
       Dispense 50p coin ;
    ENDFOR
13. FOR count GOES FROM 1 TO numberTwenties
       Dispense 20p coin ;
    ENDFOR
14. FOR count GOES FROM 1 TO numberTens
```

```
       Dispense 10p coin ;
    ENDFOR
15. FOR count GOES FROM 1 TO numberFives
       Dispense 5p coin ;
    ENDFOR
16. FOR count GOES FROM 1 TO numberTwos
       Dispense 2p coin ;
    ENDFOR
17. FOR count GOES FROM 1 to numberOnes
       Dispense 1p coin ;
    ENDFOR
```

**Using change algorithm:**

```
9. priceOfChocolateBar ← 10 ;
10. WHILE (not the end of the day)
    10.1 IF (button 1 pressed)
           IF (money > priceOfChocolateBar)
              IF (chocolateStock > 0)
                 Dispense milk chocolate ;
                 chocolateStock ← chocolateStock - 1 ;
                 Insert change algorithm here…
              ELSE
                 Display 'Sold out message' ;
              ENDIF
           ELSE
              Display 'Insufficient funds' message ;
           ENDIF
    10.2 ELSE IF (button 2 pressed)
      . . .
      . . .
      . . .
```

**Stocksfield Fire Service: Hazchem Signs**

*In this chapter you learned about alternative iteration and selection constructs. Examine your EAC algorithm and replace* `IF` *statements with* `IF...ELSE` *constructs wherever possible. What benefits does this bring?*

**Using IF ELSE**

```
// First character
IF fireFightingCode is 1
   Use coarse spray ;
```

```
ELSE IF fireFightingCode is 2
   Use fine spray ;
ELSE IF fireFightingCode is 3
   Use foam ;
ELSE
   Use dry agent ;
ENDIF
// Second character
IF precautionsCode is P
   Use LTS ;
   Dilute spillage ;
   Risk of explosion ;
ELSE IF…
…
ELSE
   Use BA & Fire kit ;
   Contain spillage ;
ENDIF
// Third character
IF character is E
   Public hazard ;
ENDIF
```

**What about dealing with an invalid code letter?**

```
// First character
IF fireFightingCode is 1
   Use coarse spray ;
ELSE IF fireFightingCode is 2
   Use fine spray ;
ELSE IF fireFightingCode is 3
   Use foam ;
ELSE IF fireFightingCode is 4
   Use dry agent ;
ELSE
   Invalid fire fighting code ;
ENDIF
// Second character
IF precautionsCode is P
   Use LTS ;
   Dilute spillage ;
   Risk of explosion ;
ELSE IF…
```

```
…
ELSE IF precautionsCode is Z
   Use BA & Fire kit ;
   Contain spillage ;
ELSE
   Invalid precautions code ;
ENDIF
// Third character
IF publicHazardCode is E
   Public hazard ;
ELSE IF publicHazardCode is blank
   No hazard ;
ELSE
   Invalid public hazard code ;
ENDIF
```

**Puzzle World: Roman Numerals and Chronograms**

*Examine your solutions to the Roman number problems and decide whether you need to use any of the alternative iteration and selection constructs. Update your algorithms accordingly.*

| Identifier | Description | Range of values |
|---|---|---|
| romanDigit | A single roman numeral | {I, V, X, L, C, D, M} |
| subSequence | a compound roman number | {IV, IX, XL, XC, CM} |
| romanNumber | a complete roman number | Many, e.g. I, LXI, MCMLXXIX, etc. |
| digitValue | value of a roman numeral | {1, 5, 10, 50, 100, 500,1000} |
| compoundValue | value of compound number | {4, 9, 50, 90, 900} |

I think we now have some of the tools necessary to expand the solution from Chapter 4. Here's where we left the algorithm for validating a roman number:

```
1.  Look at first sub-sequence in the number ;
2.  WHILE (sub-sequences to process)
       2.1.  IF current sub-sequence less than next one in the number
                2.1.2.  Display 'This number is invalid' ;
             ENDIF
       2.2.  Look at next sub-sequence ;
    ENDWHILE
```

It seems that we should be able to translate as we validate. However, to simplify the problem we could just translate a number without worrying if it's valid or not. If we

examine any roman number, e.g. MCMLXXIX we can spot an easy way to calculate its value: work backwards through the number from right to left. If the current numeral has a higher or equal value to the one we just looked at then add its value to the total otherwise subtract its value. So, working through MCMLXXIX backwards we get:
X=10, total = 10
I =1. Less than the previous X, so total = 9
X = 10. Greater than the previous I, so total = 19
X = 10. Same as previous, so total = 29
L=50. Greater than previous, so total = 79
M=1000. Greater than previous, so total = 1079.
C = 100. Less than previous, so total = 979.
M=1000. Greater than previous, so total = 1979.
Value of roman number is 1979.

We can express this algorithmically:

```
Determine length of roman number (no. of individual numerals)
total ← 0 ;
previous ← 0 ;
FOR counter GOES FROM length TO 1
   current ← numeral at position counter ;

   // Determine value of numeral
   IF (current = 'I')
      value ← 1 ;
   ELSE IF (current = 'V')
      value ← 5 ;
   ELSE IF (current = 'X')
      value ← 10 ;
   ELSE IF (current = 'L')
      value ← 50 ;
   ELSE IF (current = 'C')
      value ← 100 ;
   ELSE IF (current = 'D')
      value ← 500 ;
   ELSE
      value ← 1000 ;
   ENDIF
   IF (value < previous)
      total ← total - value ;
   ELSE
      total ← total + value ;
   ENDIF
   previous ← value ;
ENDFOR
Display total ;
```

This works nicely as long as the roman number is not malformed. To validate the number we must do a bit more work and process the number from left to right.

For any given numeral inside the number there are two possibilities: it is greater than or equal to the numeral that follows it or it is not. If it is greater or equal, then things are simple -- we just add its value to the total. If it's less than the next numeral we need to check to see if the compound it forms with the next digit is valid, i.e., we need to heed the rules we discussed in previous chapters.
Rule 1: the compound must not begin with a 'V'
Rule 2: the second numeral must be either 5 or 10 times larger than the first numeral
Rule 3: if the first numeral is not the first numeral in the whole number, then the one **before** it must be at least 10 times greater than it
Rule 4: the next but one numeral must also be less than the current one

We could define three variables `rule1`, `rule2`, `rule3`, and `rule4`. If we initialize them to 1 and then set each of them to 0 if the rule it corresponds to is met, we can then add the three variables together. If the sum is zero then the compound numeral is valid and we can translate it and add it to the total.

```
IF (numeral ≠ 'V')
   rule1 ← 0 ;
ENDIF
IF (next character value = numeral × 5) OR (next character value =
numeral × 10)
   rule2 ← 0 ;
ENDIF
IF (previous ≠ 0) AND (previous ≥ (10 × numeral))
   rule3 ← 0 ;
ENDIF
IF (next but one numeral < numeral)
   rule4 ← 0 ;
ENDIF ;
```

Using our numeral translation algorithm from the simplified solution we can put it all together:

```
previous ← 0 ;
next ← 0 ;
```

```
nextButOne ← 0 ;
counter ← 1 ;
WHILE (counter ≤ length of roman number)
   rule1 ← 0 ;
   rule2 ← 0 ;
   rule3 ← 0 ;
   rule4 ← 0 ;
   current ← character at the position given by counter ;
   // Determine value of numeral
   IF (current = 'I')
      value ← 1 ;
   ELSE IF (current = 'V')
      value ← 5 ;
   ELSE IF (current = 'X')
      value ← 10 ;
   ELSE IF (current = 'L')
      value ← 50 ;
   ELSE IF (current = 'C')
      value ← 100 ;
   ELSE IF (current = 'D')
      value ← 500 ;
   ELSE
      value ← 1000 ;
   ENDIF

   // Get value of next numeral
   IF (current not last character)
      next ← value of character at current+1 position
      // n.b. the algorithm for assigning a value to next is the
same as we just used for
      // finding the value of the current character immediately
above. I have not repeated them
      // here. Actually, what is needed is some way of taking this
repeated algorithm, storing it
      // somewhere else and using it when needed. This requires
creation of a 'function' which is
      // beyond the scope of this short book.
   ENDIF

   //Get value of next-but-one numeral
   IF (counter + 2 ≤ length of roman number) // if there is a next
but one character
      nextButOne ← value of character at current+2 position
```

```
        // Same algorithm again!
   ELSE
      nextButOne ← 0 ; // if there isn't a next but one, make the
variable zero
   ENDIF

   // Test to see if current numeral greater than or equal to the
next one
   IF (value ≥ next)
      total ← total + value ; // add it to total
   ELSE // Dealing with compound numeral

      // Establish whether the four rules have been met
      IF (value ≠ 5)
         rule1 ← 0 ;
      ENDIF
      IF (next = value × 5) OR (next  = vaulue × 10)
         rule2 ← 0 ;
      ENDIF
      IF (previous ≠ 0) AND (previous ≥ (10 × value))
         rule3 ← 0 ;
      ENDIF
      IF (nextButOne < value)
         rule4 ← 0 ;
      ENDIF ;

      // If 4 rules ARE met
      IF (rule1 + rule2 + rule3 + rule4 = 0)
         total ← total + (next - value) // e.g. if compound = IX,
we subtract current value 1 from next value 10
         counter ← counter + 2 ; // need to skip over the compound
number to the next numeral
         previous ← value ; // update previous to equal current
numeral value
      ELSE  // invalid number
         counter ← length of number + 1 ; // this will cause the
loop to terminate without processing the rest of the number
         valid ← 1 ; // show the number is invalid
      ENDIF
   ENDIF
ENDWHILE

IF (valid = 0)
```

```
   Display total ;
ELSE // invalid number
   Display 'number is invalid' ;
ENDIF
```

Note, we could remove the four rule variables by writing the following compound condition for the IF statement that uses them:

```
IF (value ≠ 5) AND
   ((next = value × 5) OR (next  = vaulue × 10)) AND
   ((previous ≠ 0) AND (previous ≥ (10 × value))) AND
   (nextButOne < value)
```

We then don't need to worry about setting and resetting `rule1`, `rule2`, `rule3`, and `rule4`. The reason I used the rule variables was to initially simplify the problem. Now that I understand how to deal with each rule I am minded to put them all together as above.

**Pangrams: Holoalphabetic Sentences**

*The pangram algorithm needs to be amended so that it allows the test to be run on any number of sentences. The algorithm should keep testing sentences until the user decides to finish. Rewrite your solution to incorporate this feature. What iteration constructs could be used? How will you solve the problem of deciding whether the user wants to continue? Explain why you used your chosen solution strategy. What other loop constructs could you have used? How would that affect the way the algorithm behaves? What other ways could you have tested to see if the user wants to finish? What are the principal advantages and disadvantages of your solution and these alternative solutions?*

```
DO
    Get sentence from user
    Start at 'A' ;
    WHILE (letters left in the alphabet)
       Write down current letter of the alphabet ;
       Move to next letter ;
    ENDWHILE
    Starting at the first letter of the sentence
    WHILE (letters left in the sentence)
       Cross off letter on the paper that matches current letter ;
       Move to next letter ;
    ENDWHILE
    Start at 'A' ;
    WHILE (letters left in the alphabet)
       IF (letter not crossed out)
          Add 1 to number of letters not crossed out ;
       ENDIF
       Move to next letter ;
    ENDWHILE
    IF (number of letters not crossed out equals zero)
       Display 'Sentence is a pangram' ;
    ENDIF
    IF (number of letters not crossed out greater than zero)
      Display 'Sentence is NOT a pangram' ;
    ENDIF
    Display 'Do you want to test another sentence? Y/N' ;
    Get userResponse ;
WHILE (userResponse ≠ 'N') ;
```

I have used a DO...WHILE loop because it works in the same sequence that a user would expect: enter a sentence then ask if another one should be processed. I could have used a WHILE loop, but then I would either have to ask the user if he/she wants to test a sentence before entering the loop for the first time (seems unnatural) or I would have to set userResponse to a default value of 'Y' before the loop. By doing this I am forcing the WHILE to iterate at least once, so I may as well use the at-least-once iteration abstraction -- the DO...WHILE.

**Online Bookstore: ISBNs**

*The ISBN validation problem is best suited to using a count-controlled loop for the part which deals with multiplying the nine digits of the number with their respective weights. Update your solution replacing the* WHILE *construct with an appropriately phrased* FOR *loop.*

*You are now in a position to tackle the hyphenation problem. For correct presentation, the ten digits of an ISBN should be divided into four parts separated by hyphens:*

- *Part 1: The country or group of countries identifier*
- *Part 2: The publisher identifier*
- *Part 3: The title identifier*
- *Part 4: The check digit*

*To keep matters as simple as possible we will only deal with hyphenating ISBNs that have a group/country code of 0 or 1 (the English language groups). The positions of the hyphens are determined by the publisher codes. To hyphenate correctly knowledge of the prefix ranges for each country or group of countries is needed. The publisher code ranges in the English group (U.S., U.K., Canada, Australia, New Zealand, etc) are given* *in Table 6.9.*

| ***Group identifier '0' publisher code ranges*** | ***If publisher ranges are between*** | ***Insert hyphens after*** | | |
|---|---|---|---|---|
| *00--19* | *00--19* | *1st digit* | *3rd digit* | *9th digit* |
| *200--699* | *20-69* | *"* | *4th "* | *"* |
| *7000---8499* | *70-84* | *"* | *5th "* | *"* |
| *85000--89999* | *85-89* | *"* | *6th "* | *"* |
| *900000---949999* | *90-94* | *"* | *7th "* | *"* |
| *9500000--9999999* | *95-99* | *"* | *8th "* | *"* |

*Using Table 6.9 develop an algorithm for displaying with correctly placed hyphens any ISBN that starts with digit 0.*

*For an extra challenge, allow ISBNs with a group code of 1 to be hyphenated. The rules for this group are slightly different than for group '0' and are given in Table 6.10.*

| ***Group identifier '1' publisher code ranges*** | ***If publisher ranges are between*** | ***Insert hyphens after*** | | |
|---|---|---|---|---|
| *00--09* | *00--09* | *1st digit* | *3rd digit* | *9th digit* |
| *100--399* | *10-39* | *"* | *4th "* | *"* |
| *4000---5499* | *40-54* | *"* | *5th "* | *"* |
| *55000--86979* | *5500-8697* | *"* | *6th "* | *"* |
| *869800---998999* | *8698-9989* | *"* | *7th "* | *"* |
| *999000--9999999* | *9990-9999* | *"* | *8th "* | *"* |

*This problem is slightly trickier than the group 0 hyphenation because you are not always just testing the first two digits of the publisher code. Use your solution to the group 0 hyphenation problem as a starting point and then work through the HTTLAP strategy to help you arrive at a solution to this problem.*

**Validation problem**

```
Display 'Enter a ten-character ISBN' ;
Get ISBN ;
total ← 0 ;
FOR counter GOES FROM 1 TO 9
   currentDigit ← character number 'counter' of ISBN ;
   total ← total + currentDigit ;
ENDFOR
checkDigit ← 10th character of ISBN ;
ENDIF
remainder ← 11 – (total ÷ 11) ;
calcCheck ← 11 – remainder ;
IF (calcCheck = 10)
```

```
   calcCheck ← 'X' ;
ENDIF
IF (calcCheck = checkDigit)
   Display ('ISBN valid') ;
ENDIF
```

**Hyphenation problem**
Here's our outline algorithm from Chapter 3:

```
1.  Determine the group code and write it out ;
2.  Write a hyphen ;
3.  Determine the publisher code and write it out ;
4.  Write a hyphen ;
5.  Determine the title code and write it out ;
6.  Write a hyphen ;
7.  Write out the check digit ;
```

We are told only to deal with group codes of 0 or 1, so determining the group code simply requires seeing whether the first character is a '0' or a '1'. All we have to do with the group code, then, is write it out followed by a hyphen. We can then use it as the basis for a top-level selection for dealing with ISBNs in either group:

```
Display 'Enter 10-character un-hyphenated ISBN' ;
currentDigit ← first character of ISBN ;
// Determine group code and write it out.
Display currentDigit ;
Display '-' ;
IF (currentDigit ='0') // Group code 0
    // Process a group 0 ISBN ;
ELSE
    // Process a group 1 ISBN ;
ENDIF ;
```

To determine the publisher code we need to use the tables above which tell us how many characters the publisher code takes depending on its first two characters. The tables also then give us the remaining hyphenation positions for once we know the length of the publisher code we also know the length of the title code: we know the group code is 1 character (we're only dealing with groups 0 and 1), we also know the check digit is 1 character, so that leaves only 8 characters for the publisher and title. However long the publisher code is, the title code must, therefore be 8-publisher code length. All we need to test the publisher code is digits 2 and 3. We could make a temporary variable called codePrefix which is the value of the second and third digits stitched together. Here goes:

```
Display 'Enter 10-character un-hyphenated ISBN' ;
currentDigit ← first character of ISBN ;
// Determine group code and write it out.
Display currentDigit ;
Display '-' ;
IF (currentDigit ='0') // Group code 0
   codePrefix ← (second digit × 10) + third digit ;
   IF (codePrefix ≤ 19)
      Display characters 2 through 3 ;
      Display '-' ;
      Display characters 4 through 9 ;
      Display '-' ;
   ELSE IF (codePrefix ≤ 69)
      Display characters 2 through 4 ;
      Display '-' ;
      Display characters 5 through 9 ;
      Display '-' ;
   ELSE IF (codePrefix ≤ 84)
      Display characters 2 through 5 ;
      Display '-' ;
      Display characters 6 through 9 ;
      Display '-' ;
   ELSE IF (codePrefix ≤ 89)
      Display characters 2 through 6 ;
      Display '-' ;
      Display characters 7 through 9 ;
      Display '-' ;
   ELSE IF (codePrefix ≤ 94)
      Display characters 2 through 7 ;
      Display '-' ;
      Display characters 8 through 9 ;
      Display '-' ;
   ELSE
      Display characters 2 through 8 ;
      Display '-' ;
      Display character 9 ;
      Display '-' ;
   ENDIF
   Display character 10 ; // the check digit
ELSE
```

```
    // Process a group 1 ISBN ;
ENDIF ;
```

Hyphenating group is slightly trickier as the publisher code depends on the value of the first two characters up to 54 but then 4 characters for the remaining codes. Actually, this is quite easy when you spot it. If you take a value such as 8899 and divide it by 100 we get the result 88 -- there are 88 hundreds in 8899. So, all we need to do is the following:

```
// Process a group 1 ISBN
codePrefix ← (second digit × 1000) + (third digit × 100) + (fourth
digit × 10) + fifth digit ;
   IF (codePrefix ÷ 100 ≤ 9)
      Display characters 2 through 3 ;
      Display '-' ;
      Display characters 4 through 9 ;
      Display '-' ;
   ELSE IF (codePrefix ÷ 100 ≤ 39)
      Display characters 2 through 4 ;
      Display '-' ;
      Display characters 5 through 9 ;
      Display '-' ;
   ELSE IF (codePrefix ÷ 100 ≤ 54)
      Display characters 2 through 5 ;
      Display '-' ;
      Display characters 6 through 9 ;
      Display '-' ;
   ELSE IF (codePrefix ≤ 8697)
      Display characters 2 through 6 ;
      Display '-' ;
      Display characters 7 through 9 ;
      Display '-' ;
   ELSE IF (codePrefix ≤ 9989)
      Display characters 2 through 7 ;
      Display '-' ;
      Display characters 8 through 9 ;
      Display '-' ;
   ELSE
      Display characters 2 through 8 ;
      Display '-' ;
      Display character 9 ;
      Display '-' ;
   ENDIF
   Display character 10 ; // the check digit
```

Now we can put the whole lot together:

```
Display 'Enter 10-character un-hyphenated ISBN' ;
currentDigit ← first character of ISBN ;
// Determine group code and write it out.
Display currentDigit ;
Display '-' ;
IF (currentDigit ='0') // Group code 0
   codePrefix ← (second digit × 10) + third digit ;
   IF (codePrefix ≤ 19)
      Display characters 2 through 3 ;
      Display '-' ;
      Display characters 4 through 9 ;
      Display '-' ;
   ELSE IF (codePrefix ≤ 69)
      Display characters 2 through 4 ;
      Display '-' ;
      Display characters 5 through 9 ;
      Display '-' ;
   ELSE IF (codePrefix ≤ 84)
      Display characters 2 through 5 ;
      Display '-' ;
      Display characters 6 through 9 ;
      Display '-' ;
   ELSE IF (codePrefix ≤ 89)
      Display characters 2 through 6 ;
      Display '-' ;
      Display characters 7 through 9 ;
      Display '-' ;
   ELSE IF (codePrefix ≤ 94)
      Display characters 2 through 7 ;
      Display '-' ;
      Display characters 8 through 9 ;
      Display '-' ;
   ELSE
      Display characters 2 through 8 ;
      Display '-' ;
      Display character 9 ;
      Display '-' ;
   ENDIF
   Display character 10 ; // the check digit
```

```
ELSE
   // Process a group 1 ISBN
   codePrefix ← (second digit × 1000) + (third digit × 100) +
(fourth digit × 10) + fifth digit ;
   IF (codePrefix ÷ 100 ≤ 9)
      Display characters 2 through 3 ;
      Display '-' ;
      Display characters 4 through 9 ;
      Display '-' ;
   ELSE IF (codePrefix ÷ 100 ≤ 39)
      Display characters 2 through 4 ;
      Display '-' ;
      Display characters 5 through 9 ;
      Display '-' ;
   ELSE IF (codePrefix ÷ 100 ≤ 54)
      Display characters 2 through 5 ;
      Display '-' ;
      Display characters 6 through 9 ;
      Display '-' ;
   ELSE IF (codePrefix ≤ 8697)
      Display characters 2 through 6 ;
      Display '-' ;
      Display characters 7 through 9 ;
      Display '-' ;
   ELSE IF (codePrefix ≤ 9989)
      Display characters 2 through 7 ;
      Display '-' ;
      Display characters 8 through 9 ;
      Display '-' ;
   ELSE
      Display characters 2 through 8 ;
      Display '-' ;
      Display character 9 ;
      Display '-' ;
   ENDIF
   Display character 10 ; // the check digit
    // Process a group 1 ISBN ;
ENDIF ;
```