

D. Debugging test

Subjects were given the same tutorial as for the listening test. Below is the documentation and source code for each of the eight exercises as it appeared in the subjects' workbooks. The source code for the exercises is given at the end of this appendix.

Exercise A1 (Trk 33 Normal speed, Trk 34 Slower)

Description

A program is required that will read in a set of student exam marks from a file and display on the screen the following:

- The highest mark scored
- The lowest mark scored
- The average mark

The exam marks are held in an input file with one student mark record per line. Each line contains the student's surname (10 character string) and the student's mark (an integer). An end of data marker (a line that contains a name of ten zeds ('ZZZZZZZZZZ') and a mark of zero (0)) terminates the file.

A pseudo-code design for the program is given below:

```

BEGIN
  Open input fileAssign (marks_file, 'marks.dat')
  Initialise lowest mark to 101
  Initialise highest mark to - 1
  Initialise sum to zero
  Initialise counter to zero
  Initialise average mark to zero
  Read first name and mark from file
  WHILE end marker not reached DO
    BEGIN
      IF mark is the lowest so far and it's > zero THEN
        Set lowest mark to mark
      IF mark is the highest so far THEN
        Set highest mark to mark
      Add mark to sum
      Add 1 to counter
      Read next name and mark from file
    END
  IF at least one mark processed (counter > zero) THEN
    Calculate average mark
    Display highest mark
    Display lowest mark
    Display average mark
  Close input file
END.

```

The input and output data are given on the next page.

Input Data

```
Swi thens  99
Wi nston   21
Hedley     67
Johnson   22
Melville   0
Lamarr     23
Severin    87
Noble      0
Spiros     68
Aherne     21
Moreton    98
Ainsley    78
Marchamp   82
Smythe     65
ZZZZZZZZZ0
```

Expected Output

```
Hi ghest mark is 99
Lowest mark is  21
Average mark is  52.21
```

Actual Output

```
Hi ghest mark is 99
Lowest mark is  21
Average mark is  52.25
```

Exercise A2 ([Trk 35 Normal speed](#), [Trk 36 Slower](#))**Description**

A program is required that will read a file of sales data and for each record report whether the transaction was excellent, average or poor. A total of all transactions is to be produced. However, it is known that SMITH made some refunds for £40 which had been agreed with head office in advance and so it is not required to take these particular refunds into account. All other refunds are to be counted when calculating the total. The input file has one record per-line of the form:

Name (10 characters) followed by Sales_value (integer)

The file is terminated by a record with a name of ZYGOTE and a sales value of zero.

Note, there may be valid sales/refunds (i.e., non-zero values) by ZYGOTE within the file.

Notice in the expected output that the refund of £40 given by SMITH (see input data) has not been subtracted from the total value of transactions (remember the refund policy described above). The program design, in the form of pseudo-code is given below:

```
BEGIN
Open input file
Initialise variables as required
Read first record
WHILE not end of data DO
    BEGIN
    IF sales value > £100 THEN
        Report 'Excellent'
    ELSE IF sales value > £50 THEN
        Report 'Average'
    ELSE IF sales value > £0 THEN
        Report 'Poor'
    ELSE
        Report 'Refund' ;
    IF record not a SMITH authorised refund THEN
        Accumulate sales total
    Accumulate counter
    Read next record
    END
Display count of transactions
Display total transaction value
Close input file
END.
```

The input and output data are given on the next page.

Input Data

```
SMITH    101
JONES    51
DAVIES   50
ZYGOTE   200
SMITH    -40
JONES    -40
SMITH    -50
JOHNSON  -10
ZYGOTE   0
```

Expected Output

```
SMITH    Excellent (101)
JONES    Average   (51)
DAVIES   Poor      (50)
ZYGOTE   Excellent (200)
SMITH    Refund    (-40)
JONES    Refund    (-40)
SMITH    Refund    (-50)
JOHNSON  Refund    (-10)
The number of transactions made was 8
The total value of transactions was 302
```

Actual Output

```
SMITH    Excellent (101)
JONES    Average   (51)
DAVIES   Poor      (50)
The number of transactions made was 3
The total value of transactions was 202
```

Exercise A3 ([Trk 37 Normal speed](#), [Trk 38 Slower](#))**Description**

A program is required that will validate a file of sales data from a shop. Each line of the input file records a sale made in the shop that day. The record contains three items:

- Quantity (integer)– the quantity sold of the product
- Price (real)– the unit price of the product
- Sales value (real)– the value of the sale

The sales value should be equal to the result of multiplying the product price by the quantity sold. For instance, if three of a product priced at £5.50 were sold, then the sales value would be $3 \times £5.50 = £16.50$. Thus, the corresponding record in the input file would be:

3 5.5 16.5

Unfortunately, the input file has been created by a sales clerk whose arithmetic is unreliable. Therefore, it is required that the program check each line of the file and report whether the record is valid (value = quantity sold \times price) or invalid (the value has been calculated incorrectly).

The pseudo-code design for the program is:

```
BEGIN
  Clear the screen
  Open the input file
  Display heading for output
  Read first record from input file
  WHILE not at end of the file DO
    BEGIN
      IF expected sales value = actual sales value THEN
        Set 'message' to 'Valid'
      ELSE
        Set 'message' to 'Invalid'
        Display record and 'message'
        Read next record
      END ;
    Close input file
  END.
```

The input and output data are given on the next page.

Input

```

2 12.99 25.00
1 6.99 6.99
3 9.99 29.97
2 9.99 19.98
3 5.00 15.00
-1 -1 -1

```

Expected Output

Quantity	Price	Value	Expected	Message
2	12.99	25.00	25.98	Invalid
1	6.99	6.99	6.99	Valid
3	9.99	29.97	29.97	Valid
2	9.99	19.98	19.98	Valid
3	5.00	15.00	15.00	Valid

Actual Output

Quantity	Price	Value	Expected	Message
2	12.99	25.00	25.98	Invalid
1	6.99	6.99	6.99	Valid
3	9.99	29.97	29.97	Invalid
2	9.99	19.98	19.98	Valid
3	5.00	15.00	15.00	Valid

Exercise A4 ([Trk 39 Normal speed](#), [Trk 40 Slower](#))**Description**

A program is required to convert lengths of various units of measurement into lengths in centimetres. An input file contains several records, one record per line. Each record contains a character (Y,I,M or C) representing yards, inches, metres and centimetres respectively. Immediately following this character is an integer representing a length. The program should display, for each record, the original measurement and the equivalent length in centimetres. To accomplish this, as each record is read in, its unit of measurement should be ascertained. If it is a Y (yards) then the length should be multiplied by the number of centimetres in a yard; if it is an I (inches) then the length needs to be multiplied by the number of centimetres in an inch; if an M (metres) then the length is multiplied by the number of centimetres in a metre; if a C (centimetres) then no conversion is made.

For example, a record of

Y2

which stands for two yards would convert to 182.88 cms.

The pseudo-code outline for this program is given below. Note, the code to 'convert length to length in centimetres' should make use of a CASE statement.

```
BEGIN
  Open input file
  WHILE not end of file DO
    BEGIN
      Read record from input file
      Convert length to length in centimetres
    END
    Display result of conversion
  END
  Close input file
END.
```

The input and output data are given on the next page.

Input Data

Y2
C5
M2
I12
M3

Expected Output

2 Y = 182.88 CMS
5 C = 5.00 CMS
2 M = 200.00 CMS
12 I = 30.48 CMS
3 M = 300.00 CMS

Actual Output

2 Y = 182.88 CMS
5 C = 5.00 CMS
2 M = 2.00 CMS
12 I = 30.48 CMS
3 M = 3.00 CMS

Exercise A5 (Trk 41 Normal speed, Trk 42 Slower)

Description

DeMorgan's laws are used in computer programming to rearrange relational expressions. If we let P and Q stand for two relational expressions (e.g. number < 3, x = y etc.) then the two laws may be expressed as:

1. NOT (P AND Q) can be re-written as (NOT P) OR (NOT Q)
2. NOT (P OR Q) can be re-written as (NOT P) AND (NOT Q)

A programmer has been experimenting with these rules and has written a program which is intended to show how a relational expression may be rearranged in various ways (in this case, four different ways). The program should read a number of values from an input file. Each line of the file contains two integer numbers, which are to be stored in the two variables, a and b. It is desired to test whether the two values both lie within the range zero to twenty inclusive. Thus, the pair of values (5,5) would generate a true result (both are between 0 and 20), whilst the following pairs of values (-1,5), (5,-1), (21,4), (4,21), (21,21), (-1,-1) would all yield a False result as in each pair, one or both of the values is outside the permitted range.

The program contains four IF... THEN statements, each of which attempts to apply this rule using various arrangements according to DeMorgan's laws. The basic condition from which these four variations have been derived is

```
IF ((a >=0) AND (a <= 20)) AND ((b >=0) AND (b <=20)) THEN
    Write ( ' Ok ' ) ;
```

Each of the four variations will display the text ' Ok ' on the screen if a true result is obtained. From examination of the output it is clear that the program is not working as expected.

The outline design is:

```
BEGIN
  Clear the screen
  Open the input file
  REPEAT
    Read a pair of values from the file
    IF first version true THEN
      Display ' Ok ' message
    IF second version true THEN
      Display ' Ok ' message
    IF third version true THEN
      Display ' Ok ' message
    IF fourth version true THEN
      Display ' Ok ' message
    Display a line of '-'
  UNTIL end of input file
  Close input_file
END.
```

The input and output data are given on the next page.

Input Data

```
0 0
1 1
5 20
21 5
-1 20
19 20
```

Expected Output

```
0k 0k 0k 0k
-----
0k 0k 0k 0k
-----
0k 0k 0k 0k
-----

-----

-----
0k 0k 0k 0k
-----
```

Actual output

```
0k 0k 0k 0k
-----
0k 0k 0k
-----
0k 0k 0k
-----

-----

0k
-----
0k 0k 0k
-----
```

Exercise A6 (Trk 43 Normal speed, Trk 44 Slower)

Description

A program is required that will sort an array of integers into ascending order. There are five integers to be sorted and they are stored in an input data file. The program should read these values, store them in an array and display the array contents on the screen. Then a variant of the *bubble sort* algorithm is to be used to sort the array elements into ascending order of value. After sorting the array contents should again be displayed on the screen.

The sorting algorithm works by considering pairs of values. If the first value of a pair is greater than the second then they should be swapped. This process is repeated for each pair of neighbouring array elements. This entire process is repeated, the number of repetitions being the number of values to be sorted (that is, 5).

The algorithm to be used can be seen in the program pseudo-code outline below.

```
BEGIN
  Clear the screen
  Open the input file
  Display a heading
  FOR each of the 5 array elements DO
    BEGIN
      Read a value from the file into array element
      Display the value
    END ;
  Write end of line
  Close input file ;
  FOR each of 5 passes of the array DO
    FOR number of array elements -1 DO
      BEGIN
        IF value of element > value of next element THEN
          BEGIN
            Store element's value in temporary variable
            Copy value of next element to this element
            Copy value of temporary variable to next element
          END
        END
      Display a heading
    FOR each element in array DO
      Display value of element
    Write end of line
  END.
```

The input and output data are given on the next page.

Input Data

1 2 5 4 3

Expected Output

Table - unsorted : (1 2 5 4 3)

Table - sorted : (1 2 3 4 5)

Actual Output

Table - unsorted : (1 2 5 4 3)

Table - sorted : (1 2 5 4 3)

Exercise A7 (Trk 45 Normal speed, Trk 46 Slower)

Description

A company has four sales representatives (and is interested in analysing the representatives' total sales performance over a period of three days. The sales data for the four representatives are stored in a data file with the figures for each representative taking up one line of the file. For each representative there is one sales figure per day, therefore, the input file has four lines with three values per line. Each value (an integer) represents a value in thousands of pounds, thus a figure of 30 means a sales value of thirty-thousand pounds.

The company policy is that a sales value of 100 or less is deemed to be *low* whilst all daily sales of **more** than 100 are *high*.

A program is required that will analyse the sales figures and display for each day the total value of all high sales and the total value of *all* low sales separately. The data are to be stored in a two-dimensional array where each of the four rows holds the three sales figures for a representative and each column represents each of the three days.

A design for the program is given below:

```
BEGIN
  Clear the screen
  Open the input file
  FOR each sales representative DO
    FOR each day DO
      Read sales figure from file into array
    Close input file
  FOR each day DO
    BEGIN
      Initialise day (column) high sales total to zero
      Initialise day (column) low sales total to zero
      FOR each representative DO
        IF sales value > 100 THEN
          Add value to high sales total
        ELSE
          Add value to low sales total
      Display high sales total
      Display low sales total
    END
  END.
```

The input and output data are given on the next page.

Input Data

```
100 100 100
20 20 20
101 90 80
200 300 400
```

Expected Output

```
High sales total for day 1 is 301
Low sales total for day 1 is 120
High sales total for day 2 is 300
Low sales total for day 2 is 210
High sales total for day 3 is 400
Low sales total for day 3 is 200
```

Actual Output

```
High sales total for day 1 is 0
Low sales total for day 1 is 320
High sales total for day 2 is 101
Low sales total for day 2 is 60
High sales total for day 3 is 301
Low sales total for day 3 is 170
```

Exercise A8 (Trk 48 Normal speed, Trk 48 Slower)**Description**

A program is required to analyse sick leave taken by employees in a company. The boss is concerned that certain employees (Fred, Jim and Paul) are taken a disproportionate number of days off sick when compared with the rest of the staff. The company has collected a data file that records sick leave. Each time an employee returns to work after being off sick, a note is made in the file of their name and the number of days for which they were absent. The file has one sickness record per line. Each line of the file contains the name of an employee (a 4 character string) and the length of the absence in days (an integer).

A program is required that will read the data in from the file and display a report which shows:

- the total number of sick days for all employees, followed by
- the number of sick days taken by the three suspects added together, followed by
- the number of sick days taken by the suspects shown as a proportion of the total days taken.

The pseudo-code design for the program is given below:

```
BEGIN
  Open input file
  Initialise total to zero
  Initialise suspect total to zero
  WHILE NOT end of input file) DO
    BEGIN
      Read a name and sick days value from file
      IF name is Fred or Jim or Paul THEN
        Add sick days value to suspect total
      Add sick days value to total
    END ;
  Display total employee absence
  Display total suspect absence
  IF total employee absence > zero THEN
    Display suspect value as proportion (i. e. suspect total divided
    by employee total)
  Close input file
END.
```

The input and output data are given on the next page.

Input Data

```
bill5  
jim 2  
sue 5  
gina1  
fred5  
jim 3  
paul8  
katy3  
fred10
```

Expected Output

```
Total employee absence = 42.00 days.  
Suspects total absence = 28.00 days.  
Suspects accounted for 0.67 of the total absence
```

Actual Output

```
Total employee absence = 42.00 days.  
Suspects total absence = 23.00 days.  
Suspects accounted for 0.55 of the total absence
```

Program Source Code

Exercise A1

```

PROGRAM Marks ;

CONST
  zeds                = 'ZZZZZZZZZZ' ;

VAR
  name                : STRING [10] ;
  mark                : Integer ;
  average             : Real ;
  highest,
  lowest,
  sum,
  counter             : Integer ;
  marks_file         : Text ;

BEGIN
  Assign (marks_file, 'marks.dat') ;
  Reset (marks_file) ;
  lowest := 101 ;
  highest := - 1 ;
  sum := 0 ;
  counter := 0 ;
  average := 0.0 ;
  Readln (marks_file, name, mark) ;
  WHILE (name <> zeds) AND (mark <> 0) DO
    BEGIN
      IF (mark < lowest) AND (mark > 0) THEN
        lowest := mark ;
      IF mark > highest THEN
        highest := mark ;
      sum := sum + mark ;
      counter := counter + 1 ;
      Readln (marks_file, name, mark) ;
    END ;
  IF counter > 0 THEN
    average := sum / counter ;
    Writeln ('Highest mark is ', highest) ;
    Writeln ('Lowest mark is ', lowest) ;
    Writeln ('Average mark is ', average:6:2) ;
    Close (marks_file) ;
  END.

```

Exercise A2

```
PROGRAM refunds ;
```

```
VAR
```

```
  name           : STRING [10] ;
  value,
  total,
  counter        : Integer ;
  datafile       : Text ;
```

```
BEGIN
```

```
  Assign (datafile, 'refunds.dat') ;
  Reset (datafile) ;
  total := 0 ;
  counter := 0 ;
  Readln (datafile, name, value) ;
  WHILE (name <> 'ZYGOTE  ') AND (value <> 0) DO
    BEGIN
      IF value > 100 THEN
        Writeln (name, ' Excellent (' , value, ')')
      ELSE IF value > 50 THEN
        Writeln (name, ' Average  (' , value, ')')
      ELSE IF value > 0 THEN
        Writeln (name, ' Poor      (' , value, ')')
      ELSE
        Writeln (name, ' Refund   (' , value, ')') ;
      IF NOT ((name = 'SMITH  ') AND (value = - 40)) THEN
        total := total + value ;
      Inc (counter) ;
      Readln (datafile, name, value) ;
    END ;
  Writeln ('The number of transactions made was ', counter) ;
  Writeln ('The total value of transactions was ', total) ;
  Close (datafile) ;
END.
```

Exercise A3

```
PROGRAM Sales ;

USES
  crt ;

VAR
  input_file      : Text ;
  price,
  sales_value     : Real ;
  message         : STRING ;
  quantity        : Integer ;

BEGIN
  ClrScr ;
  Assign (input_file, 'sales.dat') ;
  Reset (input_file) ;
  Writeln ('Quantity Price Value Expected Message') ;
  Readln (input_file, quantity, price, sales_value) ;
  WHILE (quantity <> - 1) DO
    BEGIN
      IF price * quantity = sales_value THEN
        message := 'Valid'
      ELSE
        message := 'Invalid' ;
      Writeln (quantity: 5, price: 9: 2, sales_value: 9: 2,
              price * quantity: 12: 2, message: 10) ;
      Readln (input_file, quantity, price, sales_value) ;
    END ;
  Close (input_file) ;
END.
```

Exercise A4

```
PROGRAM convert ;

CONST
  cms_per_metre    = 100 ;
  cms_per_inch     = 2.54 ;
  cms_per_yard     = 91.44 ;

VAR
  length           : Integer ;
  unit_measure     : Char ;
  display_unit     : STRING [3] ;
  length_in_cms   : Real ;
  datafile         : Text ;

BEGIN
  Assign (datafile, 'convert.dat') ;
  Reset (datafile) ;
  WHILE NOT Eof (datafile) DO
    BEGIN
      Readln (datafile, unit_measure, length) ;
      CASE unit_measure OF
        'I'       : length_in_cms := length * cms_per_inch ;
        'm'       : length_in_cms := length * cms_per_metre ;
        'Y'       : length_in_cms := length * cms_per_yard ;
        ELSE      : length_in_cms := length ;
      END ;
      Writeln (length:3, unit_measure:2, ' = ', length_in_cms:7:2, ' CMS') ;
    END ;
  Close (datafile) ;
END.
```

Exercise A5

```
PROGRAM DeMorgan ;

USES
  Crt ;

VAR
  a,
  b          : Integer ;
  input_file : Text ;

BEGIN
  ClrScr ;
  Assign (input_file, 'DeMorgan.dat') ;
  Reset (input_file) ;
  REPEAT
    Readln (input_file, a, b) ;
    IF NOT ((a < 0) OR (a > 20)) AND NOT ((b < 0) OR (b > 20)) THEN
      Write (' Ok ') ;
    IF NOT (NOT (NOT ((a < 0) OR (a > 20))) OR NOT (NOT ((b < 0) OR (b > 20)
    ))) THEN
      Write (' Ok ') ;
    IF NOT (NOT (a <= 0) OR (a >= 21)) AND NOT ((b < 0) OR (b >= 21)) THEN
      Write (' Ok ') ;
    IF NOT ((a <= - 1) OR (a > 20)) AND NOT ((b < 0) OR (b > 20)) THEN
      Write (' Ok ') ;
    Writeln ;
    Writeln ('-----') ;
  UNTIL Eof (input_file) ;
  Close (input_file) ;
END.
```

Exercise A6

```

PROGRAM Sort ;

USES
  Crt ;

CONST
  n          = 5 ;

VAR
  i,
  j,
  temp      : Integer ;
  table     : ARRAY [1 .. n] OF Integer ;
  sorted    : Boolean ;
  input_file : Text ;

BEGIN
  ClrScr ;
  Assign (input_file, 'sort.dat') ;
  Reset (input_file) ;
  Write ('Table - unsorted : (') ;
  FOR i := 1 TO n DO
    BEGIN
      Read (input_file, table [i]) ;
      Write (table [i] : 2, ' ') ;
    END ;
  Writeln (')') ;
  Close (input_file) ;
  FOR i := 1 TO n DO
    FOR j := 1 TO n - 1 DO
      BEGIN
        IF (table [j] > table [j + 1]) THEN
          BEGIN
            temp := table [j] ;
            table [j] := table [j + 1] ;
            table [j + 1] := temp ;
          END ;
        END ;
      Write ('Table - sorted : (') ;
      FOR i := 1 TO n DO
        Write (table [i] : 2, ' ') ;
      Writeln (')') ;
    END.

```

Exercise A7

```
PROGRAM Table ;

USES
  crt ;

VAR
  i,
  j          : Integer ;
  numbers    : ARRAY [1..4, 1..3] OF Integer ;
  input_file : Text ;
  column_hi_total,
  column_low_total : Integer ;

BEGIN
  ClrScr ;
  Assign (input_file, 'table.dat') ;
  Reset (input_file) ;
  FOR i := 1 TO 4 DO
    FOR j := 1 TO 3 DO
      Read (input_file, numbers [i, j]) ;
    Close (input_file) ;
  FOR j := 1 TO 3 DO
    BEGIN
      column_hi_total := 0 ;
      column_low_total := 0 ;
      FOR i := 1 TO 4 DO
        IF numbers [j, i] > 100 THEN
          column_hi_total := column_hi_total + numbers [j, i]
        ELSE
          column_low_total := column_low_total + numbers [j, i] ;
      Writeln ('High sales total for day ', j, ' is ', column_hi_total) ;
      Writeln ('Low sales total for day ', j, ' is ', column_low_total) ;
    END ;
  END.
```

Exercise A8

```
PROGRAM illness ;

VAR
  name           : STRING [4] ;
  no_days,
  total,
  suspect_total  : Real ;
  datafile       : Text ;

BEGIN
  Assign (datafile, 'illness.dat') ;
  Reset (datafile) ;
  total := 0 ;
  suspect_total := 0 ;
  WHILE NOT Eof (datafile) DO
    BEGIN
      Readln (datafile, name, no_days) ;
      IF (name = 'fred') OR (name = 'jim') OR (name = 'paul') THEN
        suspect_total := suspect_total + no_days ;
        total := total + no_days ;
      END ;
    Writeln ('Total employee absence = ', total:6:2, ' days.') ;
    Writeln ('Suspects total absence = ', suspect_total:6:2, ' days.') ;
    IF total > 0 THEN
      Writeln ('Suspects accounted for ', suspect_total / total:4:2,
        ' of the total absence') ;
    Close (datafile) ;
  END.
```