Introduction to Flowcharts

This section covers the use of flow diagrams (charts) in the production of algorithms.

This section primarily covers four areas:

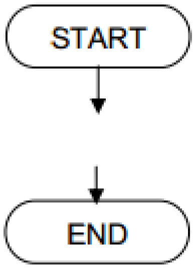
1 Common flow chart symbols

2 Writing flowcharts to solve problems

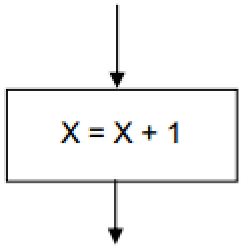
3 Dry running of flowcharts to determine its function and outputs

4 Exercises to test the above concepts

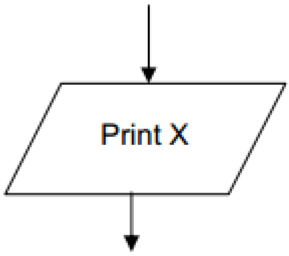
1.1 The start and end box:



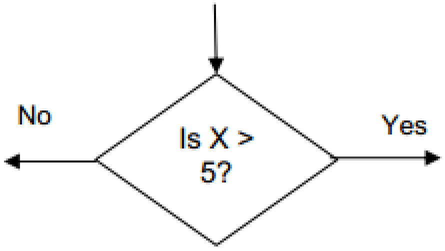
1.2 The process box:



1.3 Input/Output box:



1.4 Decision/query box



1.1 WRITING FLOWCHARTS TO SOLVE PROBLEMS

The following five problems are also covered in section 3.2 where the algorithms are constructed using pseudocode. Candidates may choose to answer questions using either flowcharts or pseudocode but a working knowledge of both techniques is well advised.

Example 1

A town contains 5000 houses. Each house owner must pay tax based on the value of the house. Houses over $200 000 pay 2% of their value in tax, houses over $100 000 pay 1.5% of their value in tax and houses over $50 000 pay 1% of their value in tax. All others pay no tax. Write an algorithm to solve this problem in the form of a flowchart.

Example 2

The following formula is used to calculate n: n = (x * x)/(1 – x). The value x = 0 is used to stop the algorithm. The calculation is repeated using values of x until the value x = 0 is input. There is also a need to check for error conditions. The values of n and x should be output. Write an algorithm to show this repeated calculation in the form of a flowchart.

Example 3

Write an algorithm in the form of a flowchart which takes temperatures input over a 100 day period (once per day) and outputs the number of days when the temperature was below 20C and the number of days when the temperature was 20C and above.

Example 4

Write an algorithm in the form of a flowchart which:

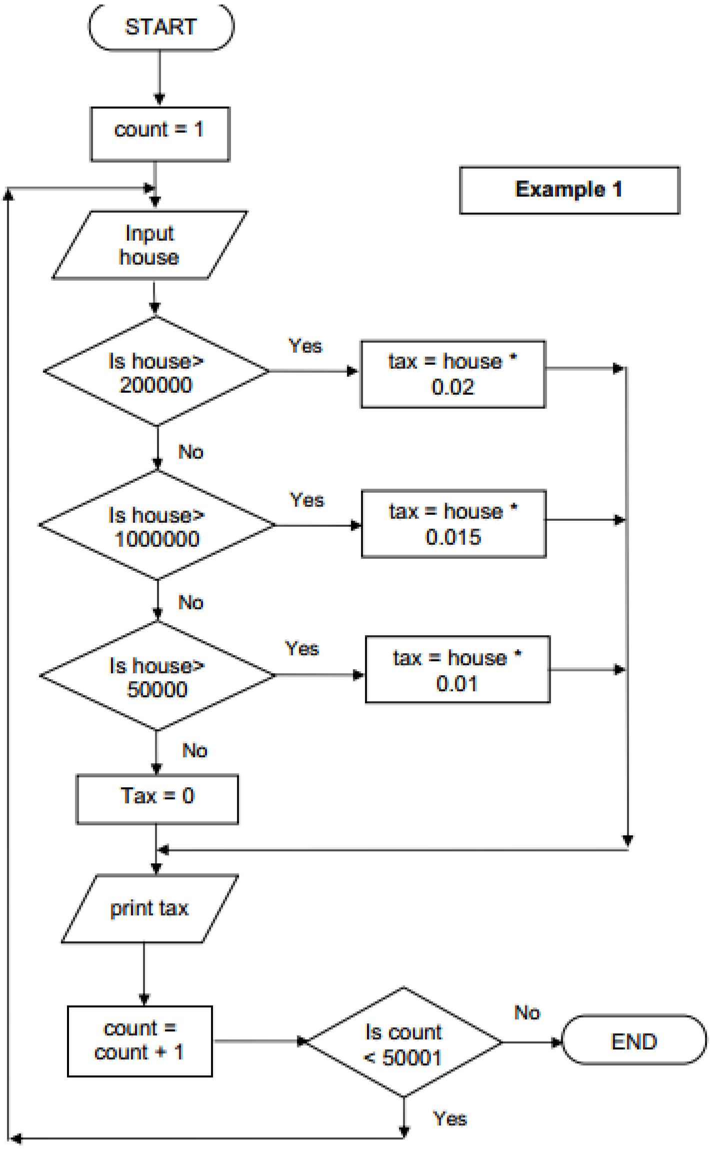
- inputs the top speeds (in km/hr) of 5000 cars
- outputs the fastest speed and the slowest speed
- outputs the average (mean) speed of all the 5000 cars

Example 5

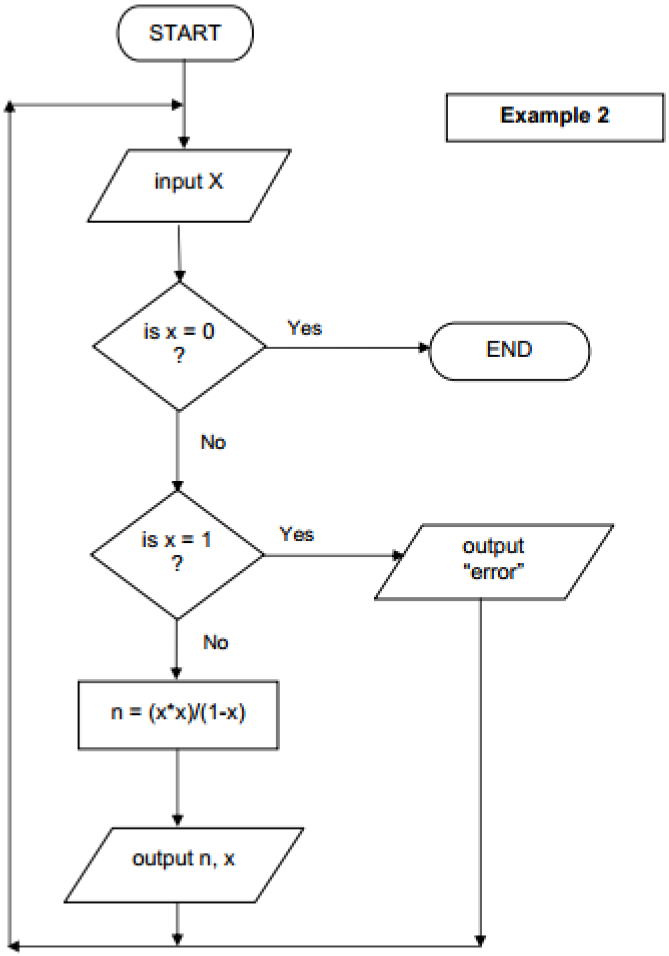
A shop sells books, maps and magazines. Each item is identified by a unique 4 – digit code. All books have a code starting with 1, all maps have a code starting with 2 and all magazines have a code starting with 3. The code 9999 is used to end the algorithm.

Write an algorithm in the form of a flowchart which inputs the codes for all items in stock and outputs the number of books, number of maps and the number of magazines in stock. Include any validation checks needed.

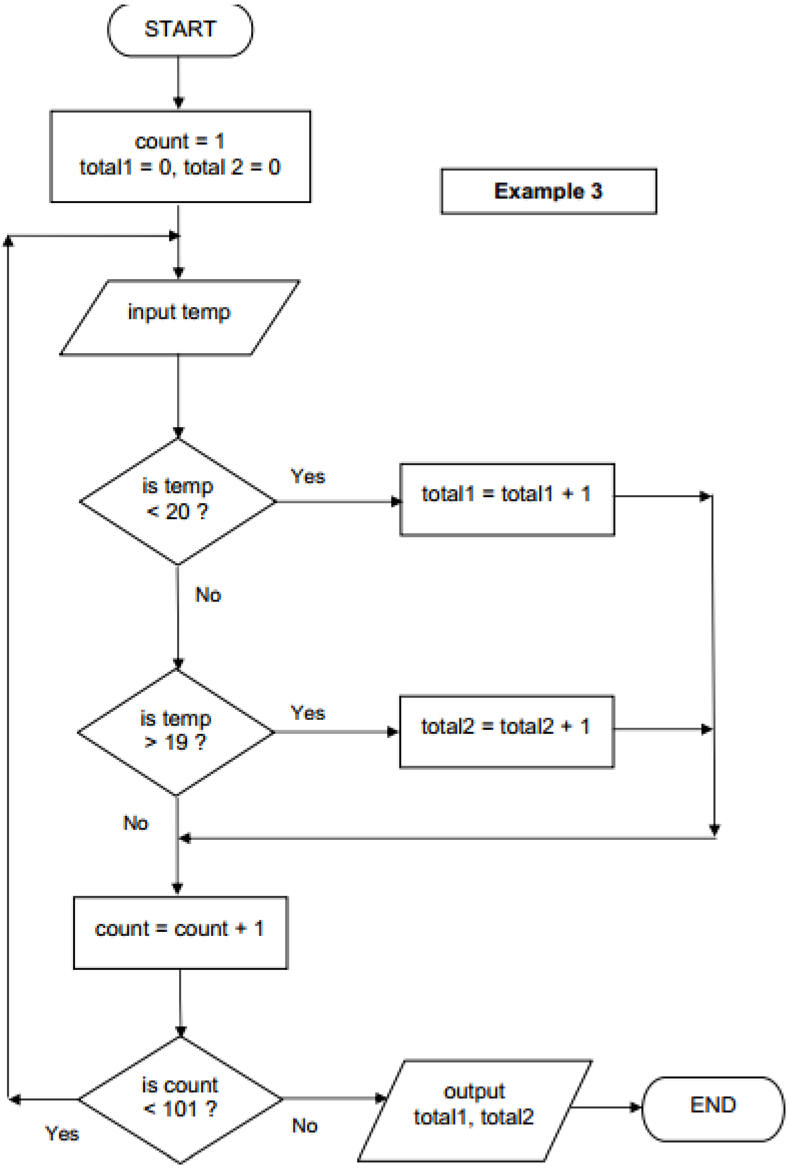
Answer 1


START
count = 1
Example 1
Input house
Is house> 200000
Yes
tax = house * 0.02
No
Is house> 1000000
Yes
tax = house * 0.015
No
Is house> 50000
Yes
tax = house * 0.01
No
Tax = 0
print tax
count = count + 1
Is count < 50001
No
END
Yes

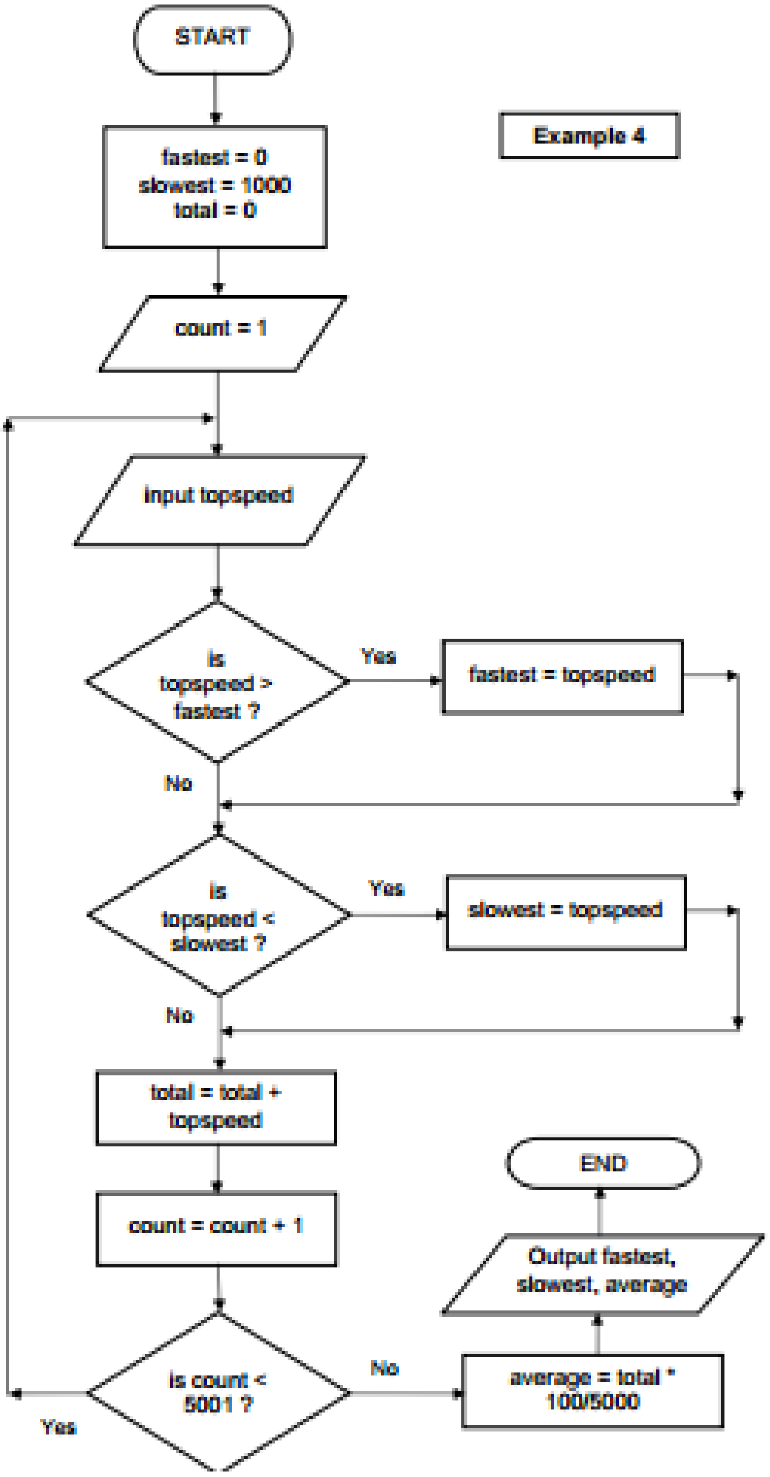
Answer 2


START
Example 2
input X
is x = 0
?
Yes
END
No
is x = 1
?
Yes
output
"error"
No
n = (x*x)/(1-x)
output n, x

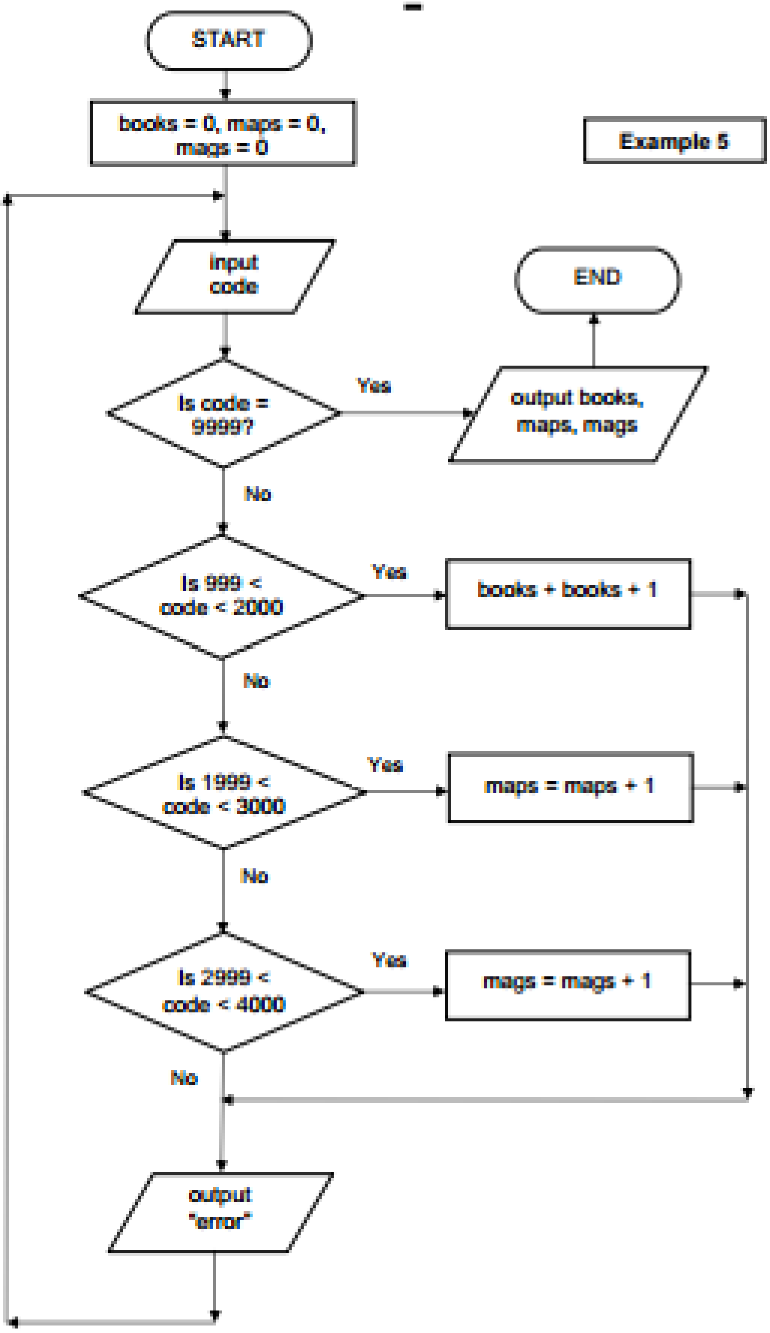
Answer 3


START
count = 1
total1 = 0, total 2 = 0
Example 3
input temp
is temp
< 20 ?
Yes
total1 = total1 + 1
No
is temp
> 19 ?
Yes
total2 = total2 + 1
No
count = count + 1
is count
< 101 ?
Yes
No
output
total1, total2
END

Answer 4


START
fastest = 0
slowest = 1000
total = 0
count = 1
input topspeed
is topspeed > fastest ?
Yes
fastest = topspeed
No
is topspeed < slowest ?
Yes
slowest = topspeed
No
total = total + topspeed
count = count + 1
is count < 5001 ?
Yes
No
average = total * 100/5000
Output fastest, slowest, average
END
Example 4

Answer 5


START
books = 0, maps = 0,
mags = 0
Example 5
input
code
END
Is code =
9999?
Yes
output books,
maps, mags
No
Is 999 <
code < 2000
Yes
books + books + 1
No
Is 1999 <
code < 3000
Yes
maps = maps + 1
No
Is 2999 <
code < 4000
Yes
mags = mags + 1
No
output
"error"

1.2 DRY RUNNING OF FLOWCHARTS

Dry running of flowcharts is basically a technique to:

- determine the output for a known set of data to check it carries out the task correctly
- check on the logic of the algorithm
- determine the function of the algorithm

When dry running a flowchart it is advisable to draw up a trace table showing how variables change their values at each stage in the algorithm. The advantages of doing this are:

- if you make a mistake, it is easier to back track to where the error occurred rather than starting from the beginning again
- there is less chance of an error being made
- encourages a more logical approach

The following three examples show all stages in the dry running for the given set of input data:

Example 1

This algorithm inputs 3 numbers, each number goes through successive division by 10 until its value is less than 1. An output is produced which contains the number input and a value generated by the flowchart processing.

Data to be used: X = 85, 3190, -40

Example 2

This algorithm inputs 5 values and outputs how many input numbers were negative and how many were positive.

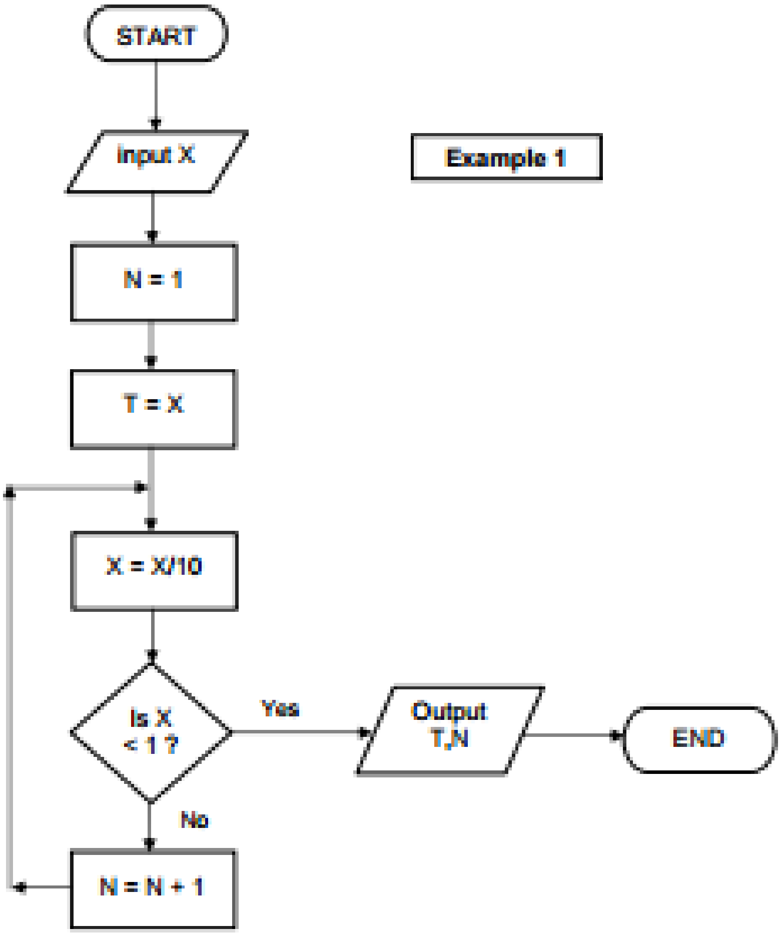
Data to be used: N = 1, -5, 2, -8, -7

Example 3

This algorithm inputs the number of hours of sunshine recorded each day for a week (7 days). The output is the highest value for hours of sunshine and the average (mean) value for the numbers of hours of sunshine per day.

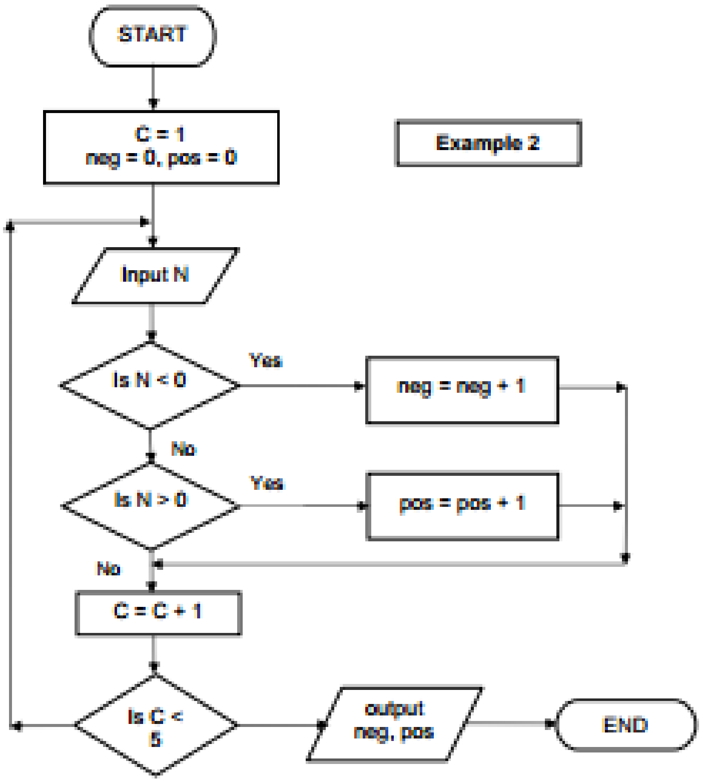
Data to be used: hours = 9.0, 7.8, 1.2, 4.5, 10.0, 6.4, 3.1

Example 1



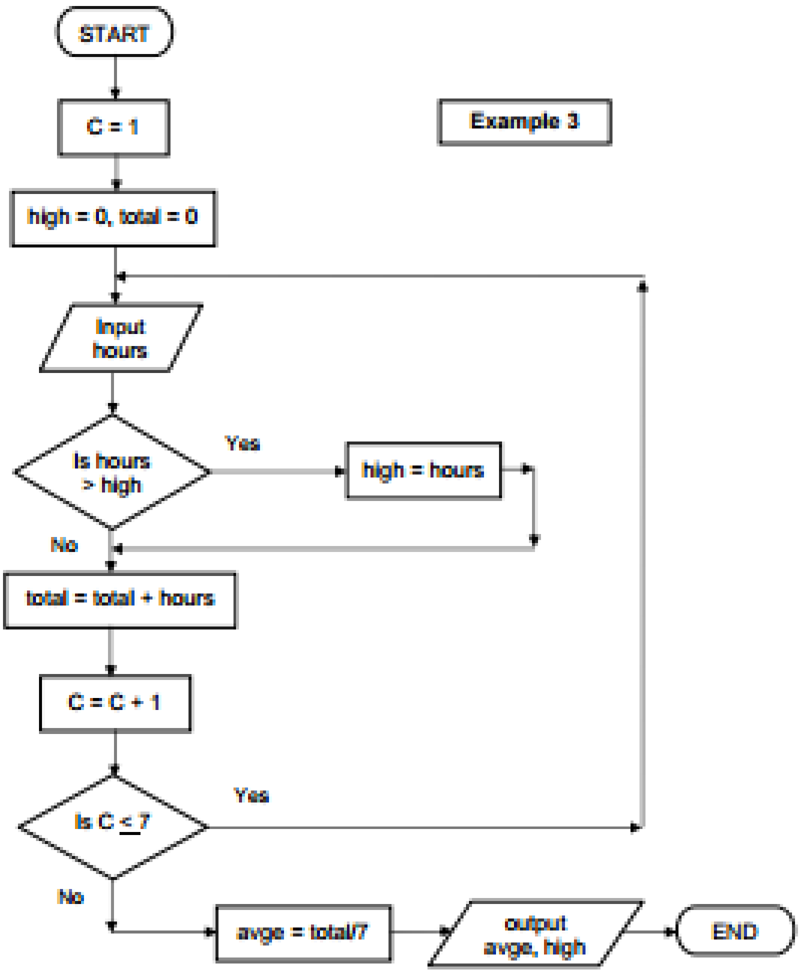
Trace Table

X	N	T	Output T, N
85	1	85	
8.5	2		
0.85			85, 2
3190	1	3190	
319	2		
31.9	3		
3.19	4		
0.319			3190, 4
-40	1	-40	
-4			-40, 1



Trace Table

C	N	neg	pos	Output neg, pos
1	1	0	0	
2	-5	1	1	
3	2	2	2	
4	-8	3		
5	-7			
6				3, 2



Trace Table

C	hours	high	total	avge	Output avge, high
1	9	0	0	6	
2	7.8	9	9		
3	1.2	10	16.8		
4	4.5		18		
5	10		22.5		
6	6.4		32.5		
7	3.1		38.9		
8			42		
					6, 10

1.3INTRODUCTION TP PSEUDOCODE

This section covers the use of pseudocode in the production of algorithms.
Candidates should use standard computing text books to find out information on the features of programming languages (high level and low level), interpreters, compilers, assemblers, user documentation and technical documentation.
No specific programming language is referred to; development of algorithms using pseudocode uses generic descriptions of looping, branching, data manipulation, input/output, totaling and counting techniques.

The section is broken down into four areas:
1 description of common pseudocode terms
2 writing algorithms using pseudocode
3 finding errors in sections of pseudocode
4 exercises

Common pseudocode terms

- **counting**

Counting in 1s is quite simple; use of the statement count = count + 1 will enable counting to be done (e.g. in controlling a repeat loop). The statement literally means: the (new) count = the (old) count + 1

It is possible to count in any increments just by altering the numerical value in the statement (e.g. count = count – 1 counts backwards)

- **totalling**

To add up a series numbers the following type of statement should be used:

total = total + number

This literally means (new) total = (old) total + value of number

- input/output

Input and output indicated by the use of the terms input number, output total,

print total, print “result is” x and so on.

- **branching**

There are two common ways of branching:

case of ….. otherwise …... endcase

if ….. then ….. else ….. endif

```
case of

case number of
1: x = x + 1
2: y = y + 1
otherwise print "error"
endcase
```

```
if … then

if number = 1 then x = x + 1
    else if number = 2 then y = y + 1
        else print "error"
    endif
endif
```

- **loops**

There are three common ways of performing a looping function:

for … to … next, while … endwhile and repeat … until

The following example input 100 numbers and finds the total of the 100 numbers and outputs this total. All three looping techniques are shown:

for … to

```
for count = 1 to 100
input number
total = total + number
next
print total
print total
```

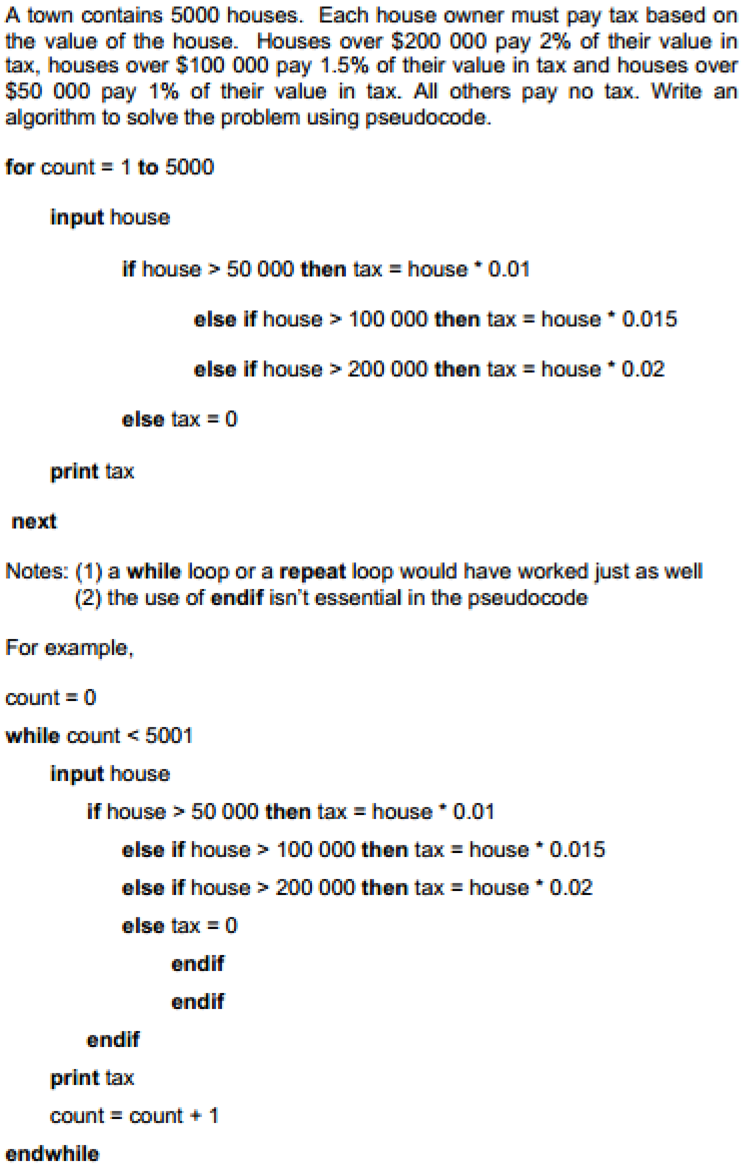
while … endwhile

```
while count < 101
input number
total = total + number
count = count + 1
endwhile
print tot
```

repeat … until

```
repeat
input number
total = total + number
count = count + 1
until count = 100
```

1.4 WRITING ALGORITHMS USING PSEUDOCODE



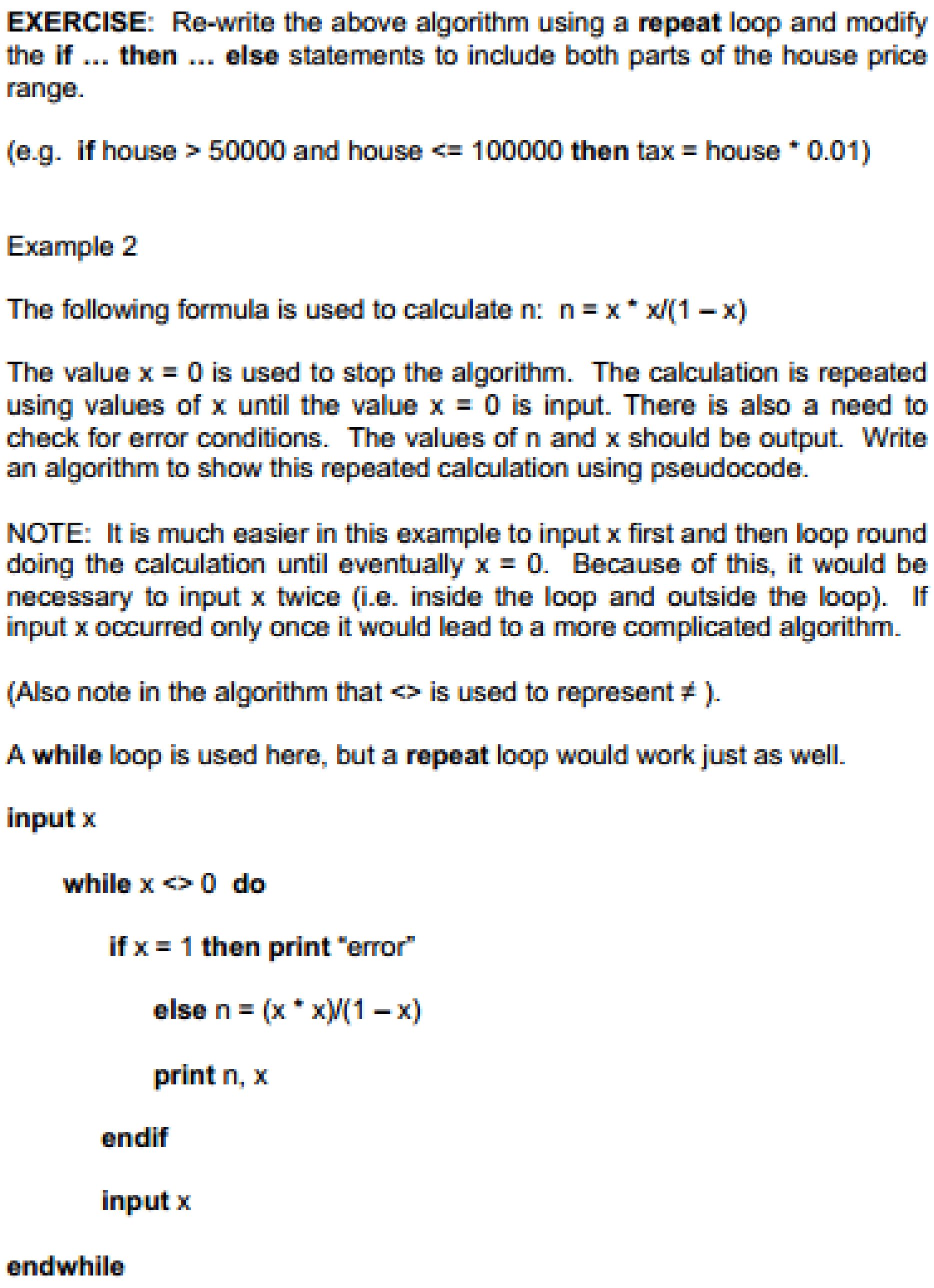
A town contains 5000 houses. Each house owner must pay tax based on the value of the house. Houses over $200 000 pay 2% of their value in tax, houses over $100 000 pay 1.5% of their value in tax and houses over $50 000 pay 1% of their value in tax. All others pay no tax. Write an algorithm to solve the problem using pseudocode.

```
for count = 1 to 5000

      input house

               if house > 50 000 then tax = house * 0.01

                        else if house > 100 000 then tax = house * 0.015

                        else if house > 200 000 then tax = house * 0.02

               else tax = 0

      print tax

 next
```

Notes: (1) a **while** loop or a **repeat** loop would have worked just as well
(2) the use of **endif** isn't essential in the pseudocode

For example,

```
count = 0
while count < 5001
      input house
           if house > 50 000 then tax = house * 0.01
               else if house > 100 000 then tax = house * 0.015
               else if house > 200 000 then tax = house * 0.02
               else tax = 0
                     endif
                     endif
           endif
      print tax
      count = count + 1
endwhile
```



EXERCISE: Re-write the above algorithm using a **repeat** loop and modify the **if … then … else** statements to include both parts of the house price range.

(e.g. **if** house > 50000 and house <= 100000 **then** tax = house * 0.01)

Example 2

The following formula is used to calculate n: n = x * x/(1 – x)

The value x = 0 is used to stop the algorithm. The calculation is repeated using values of x until the value x = 0 is input. There is also a need to check for error conditions. The values of n and x should be output. Write an algorithm to show this repeated calculation using pseudocode.

NOTE: It is much easier in this example to input x first and then loop round doing the calculation until eventually x = 0. Because of this, it would be necessary to input x twice (i.e. inside the loop and outside the loop). If input x occurred only once it would lead to a more complicated algorithm.

(Also note in the algorithm that <> is used to represent ≠).

A **while** loop is used here, but a **repeat** loop would work just as well.

```
input x
    while x <> 0  do
        if x = 1 then print "error"
            else n = (x * x)/(1 – x)
            print n, x
       endif
       input x
endwhile
```

Example 3

Write an algorithm using pseudocode which takes temperatures input over a 100 day period (once per day) and output the number of days when the temperature was below 20C and the number of days when the temperature was 20C or above.

(NOTE: since the number of inputs is known, a **for … to** loop can be used. However, a **while** loop or a **repeat** loop would work just as well).

```
total1 = 0: total2 = 0

for days = 1 to 100

    input temperature

         if temperature < 20 then total1 = total1 + 1

              else total2 = total2 + 1

         endif

next

print total1, total2
```

This is a good example of an algorithm that could be written using the **case** construct rather than **if … then … else**. The following section of code replaces the statements ***if*** *temperature < 20* ***then …… endif****:*

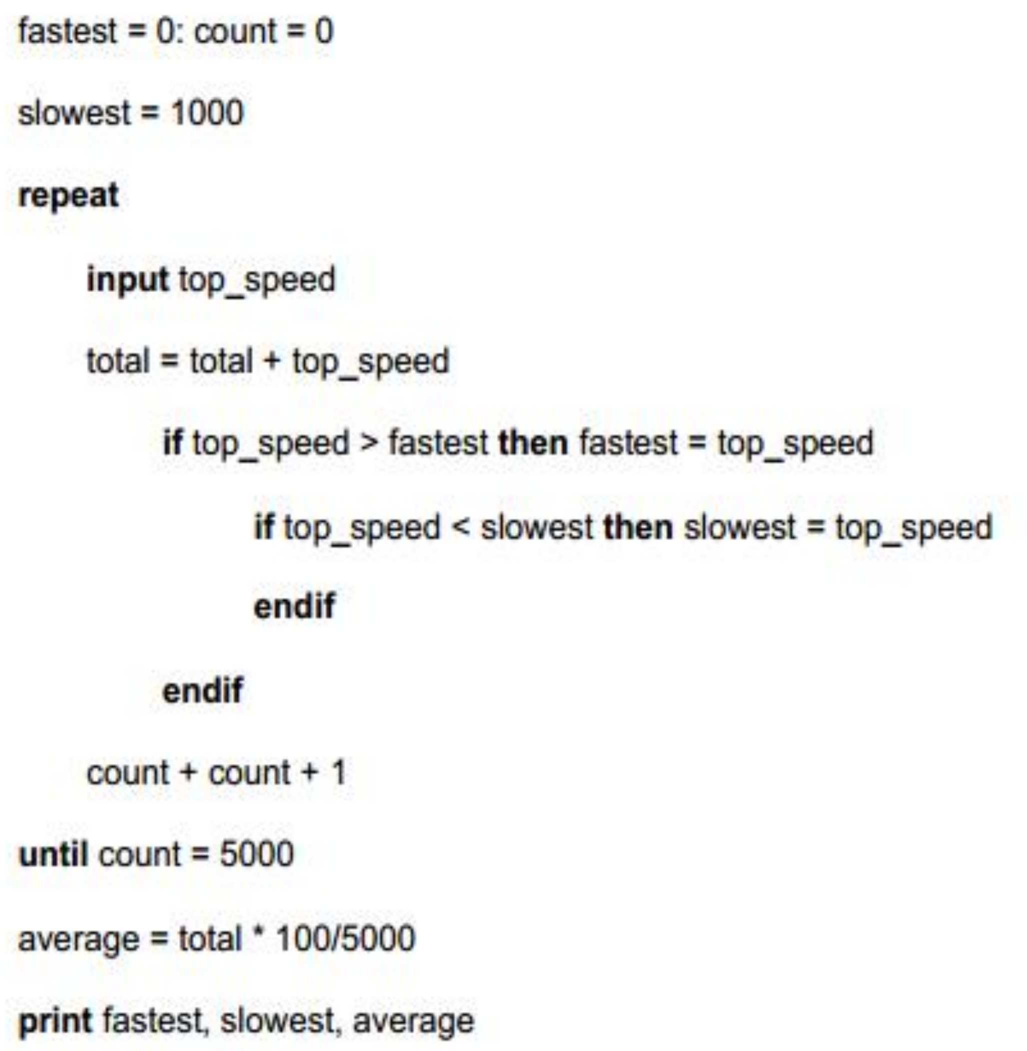
```
        case temperature of

             1:  total1 = total1 + 1

             2:  total2 = total2 + 1

        endcase
```

Example 4

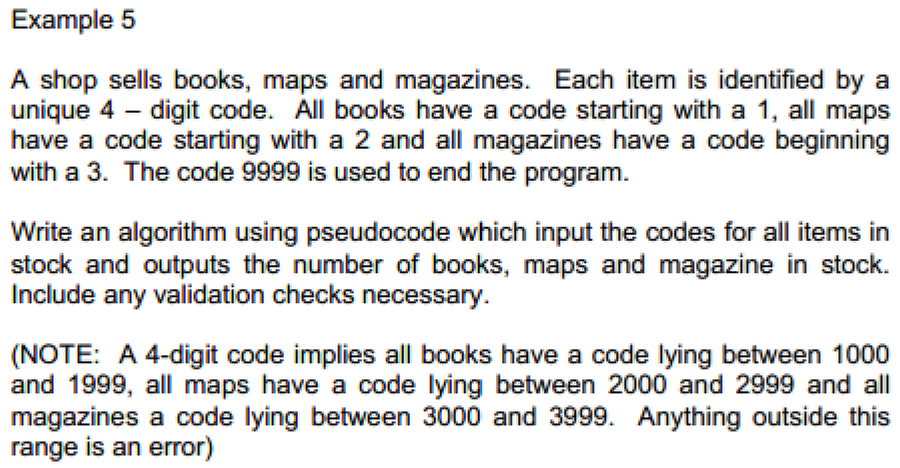
Write an algorithm using pseudocode which:

- inputs the top speeds of 5000 cars
- outputs the fastest speed and the slowest speed
- outputs the average speed of all the 5000 cars

(NOTE: Again since the actual number of data items to be input is known any one of the three loop structures could be used. It is necessary to set values for the fastest (usually set at zero) and the slowest (usually set at an unusually high value) so that each input can be compared. Every time a value is input which > the value stored in fastest then this input value replaces the existing value in fastest; and similarly for slowest).



```
fastest = 0: count = 0
slowest = 1000
repeat
    input top_speed
    total = total + top_speed
        if top_speed > fastest then fastest = top_speed
            if top_speed < slowest then slowest = top_speed
            endif
        endif
    count + count + 1
until count = 5000
average = total * 100/5000
print fastest, slowest, average
```

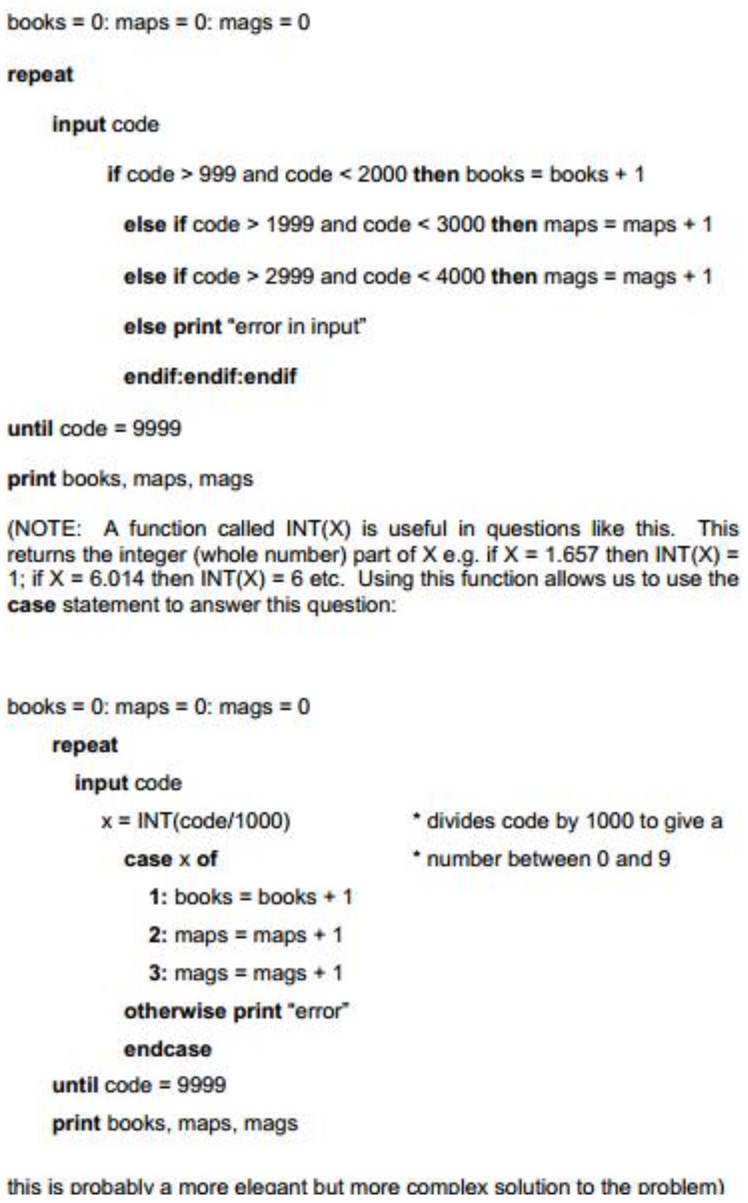


Example 5

A shop sells books, maps and magazines. Each item is identified by a unique 4 – digit code. All books have a code starting with a 1, all maps have a code starting with a 2 and all magazines have a code beginning with a 3. The code 9999 is used to end the program.

Write an algorithm using pseudocode which input the codes for all items in stock and outputs the number of books, maps and magazine in stock. Include any validation checks necessary.

(NOTE: A 4-digit code implies all books have a code lying between 1000 and 1999, all maps have a code lying between 2000 and 2999 and all magazines a code lying between 3000 and 3999. Anything outside this range is an error)



```
books = 0: maps = 0: mags = 0

repeat

    input code

        if code > 999 and code < 2000 then books = books + 1

          else if code > 1999 and code < 3000 then maps = maps + 1

          else if code > 2999 and code < 4000 then mags = mags + 1

          else print "error in input"

          endif:endif:endif

until code = 9999

print books, maps, mags
```

(NOTE: A function called INT(X) is useful in questions like this. This returns the integer (whole number) part of X e.g. if X = 1.657 then INT(X) = 1; if X = 6.014 then INT(X) = 6 etc. Using this function allows us to use the **case** statement to answer this question:

```
books = 0: maps = 0: mags = 0
    repeat
      input code
        x = INT(code/1000)            * divides code by 1000 to give a
          case x of                   * number between 0 and 9
            1: books = books + 1
            2: maps = maps + 1
            3: mags = mags + 1
          otherwise print "error"
          endcase
    until code = 9999
    print books, maps, mags
```

this is probably a more elegant but more complex solution to the problem)