# Worksheet 2 Structured programming Answers

**Task 1**

1. The following program is written as it would have been before the days of structured programming.

It is designed to allow a user to input the times taken, to the nearest minute, by different people to complete a certain task. The program outputs the number of people who took

* more than 60 minutes
* between 30 and 59 minutes
* less than 30 minutes

slow = 0

medium = 0

fast = 0

INPUTDATA

timeTaken = USERINPUT

IF timeTaken = 0 GOTO PRINT

IF timeTaken < 30 GOTO UNDER30

IF timeTaken <60 GOTO UNDER60

slow = slow + 1

GOTO INPUTDATA

UNDER30

fast = fast + 1

GOTO INPUTDATA

UNDER60

medium = medium + 1

GOTO INPUTDATA

PRINT

OUTPUT fast, medium, slow

(i) Rewrite the programming using structured programming techniques.

slow = 0

medium = 0

fast = 0

timeTaken = USERINPUT

WHILE timeTaken <> 0

IF timeTaken < 30 THEN

fast = fast + 1

ELSE

IF timeTaken < 60 THEN

medium = medium + 1

ELSE

slow = slow + 1

ENDIF

timeTaken = USERINPUT

ENDWHILE

OUTPUT fast, medium, slow

(ii) Flowcharts were invented in the days of GOTO statements. That is why they are not well-suited to representing iteration and selection structures. Can you draw a flowchart of the unstructured program?

Start

Input timeTaken

Yes

timeTaken

= 0?

Output totals

End

No

Yes

timeTaken

< 30?

fast = fast + 1

No

Yes

timeTaken

< 60?

medium = medium + 1

No

slow = slow + 1

(iii) Which version of the program is

* quicker to write?
* easier to understand?
* less likely to contain errors?

The structured version for all three!

(iv) Another feature of some early programming languages was that no identifier (e.g. variable name or label) could be more than 6 characters. How would this affect program readability, ease of debugging and maintenance?

This is intended to make the point that meaningful variable names are very important for readability…

2. (a) MOD is an arithmetic operator which returns the remainder from integer division.

e.g. x = 27 MOD 4 will put the value 3 in x.

DIV performs integer division.

e.g. y = 27 DIV 4 will put the value 6 in y.

Write pseudocode statements to allow the user to input a 3-digit number, and then print out the individual digits in the number.

e.g. If the user enters 465, the output should be “the digits are 4 6 5

i🡨USERINPUT

hundreds = i DIV 100

units = i MOD 10

j = i DIV 10

tens = j MOD 10

OUTPUT (“The digits are “, hundreds, tens, units)

see Python program **worksheet 2 Question 2.py**

(b) Devise a pseudocode algorithm which generates and prints all 3-digit numbers that equal the sum of the cubes of their individual digits.

e.g. 153 satisfies this condition because 153 = 13 + 53 + 33

(In pseudocode, express this as 1\*\*3 + 5\*\*3 + 3\*\*3)

FOR i =101 TO 999

hundreds = i DIV 100

units = i MOD 10

j = i DIV 10

tens = j MOD 10

sumOfCubes = hundreds\*\*3 + tens\*\*3 + units\*\*3

IF i = sumOfCubes

print("number is ", hundreds, tens, units)

ENDIF

ENDFOR

see Python program **sumOfCubes.py**

**Task 2**

3. A hierarchy chart can be compared to an upside-down tree, with the root at the top and branches and leaves spreading downwards.

The “leaves” are the lowest level modules and all or most of the detailed program code will be in the “leaves”.

In the hierarchy chart below:

(a) Which are the Level 1 modules? 2, 3, 4, 5

(b) Which are the Level 2 modules? 6, 7 ,8, 9

(c) Which are the Level 3 modules? 10, 11

(d) Write down the order in which the modules are executed. 6 7 3 10 11 9 5

**6**

**1**

**3**

**2**

**4**

**7**

**9**

**8**

**11**

**10**

**5**

4. What are the advantages of structured programming?

* Each module can be individually tested
* Modules can be re-used several times in a program
* Frequently used modules can be saved in a library and used by other programs
* Several programmers can simultaneously work on different modules, shortening development time
* Structured programs are easier to understand and maintain

5. The following pseudocode program is designed to allow the user to input a series of three numbers and for each set of numbers, find and output the maximum. The maximum is then added to a total. When the user enters 000 for the three numbers, the average of all the maximums is calculated and output.

**Averages**

**Prompt and input numbers**

**Find maximum**

**Initialise**

**Process data**

**Calculate and print average**

**Perform calculations**

**Level 1**

**Level 2**

see Python programs Worksheet 2 maximums.py

worksheet 2 max as subs.py