Topic: 2.2.1 Programming concepts

Declare and use variables and constants

Variable
A variable is a value that can change during the execution of a program.

Constant
A constant is a value that is set when the program initializes and does not change throughout the program's execution.

Declaration
A declaration is a statement in a program that gives the compiler or the interpreter information about a variable or constant that is to be used within a program.
A declaration ensures that sufficient memory is reserved in which to store the values and also states the variables' data type. Reserved words/keywords cannot be used as identifier names as this generates a compiler error and comes under the category of syntax error.

Declaration of local variables

- DIM MyCounter AS Integer
- DIM FirstName, LastName AS String (declares two variables)
- DIM TheLength AS Single
- DIM DOB AS Date
- DIM OverDueFlag AS Boolean

Scope
Scope indicates whether a variable can be used by all parts of a program or only within limited sections of the program – for example, within a subroutine.

Global variable
A global variable is one that is declared at the start of the main program and is visible (useable) everywhere in the program and exists for the lifetime of the program.
Note that if one procedure changes the value of a global variable, then the next procedure that uses the variable will be given this changed value – this is a common cause of errors.

Local variable
A local variable is one that is only visible inside the procedure or function in which it is declared.

Note that a local variable cannot be referenced from outside the procedure. In fact a local variable does not exist until the procedure starts executing and it disappears when the procedure stops executing. Thus any value that is held by a local variable is only stored temporarily.
The lifetime of a local variable is the lifetime of the procedure in which the local variable is declared.
The advantage of using local variables rather than global variables is that the same variable names can be used in several different procedures without any chance of values being confused.

Topic: 2.2.1 Programming concepts

Features of variables:

- Holds data in memory
- Are assigned a data type
- The data can change throughout the program's operation
- The data input must be the same data type as assigned to that variable

Variables are assigned a name that holds the data. That name is used to distinguish it from the other variables in your program. The name must not start with a number or character that is not a letter. Also, the name of the variable must not be a reserved word like PRINT, INPUT, LET, ABS, BEEP, etc.

There are two ways to declare a variable in Basic.

The first is to put a data type symbol after the name

$	String
%	Integer
!	Single
#	Double

Examples:

MyName$
Num1%
Num2!
Answer!

Sample Code:

```
CLS
Header$ = "This is an example program"
Num1% = 5
Num2% = 6
Num5! = 4.5
Num6! = 6.75
Num7# = 56000.25
Num8# = 89000.34
CLS
PRINT Header$
PRINT Num1% + Num2%
PRINT Num6! / Num5!
```

Topic: 2.2.1 Programming concepts

```
PRINT Num8# + Num2%
```

The second way is the preferred way since Visual Basic uses this method. DIM is used to make variables of a data type.

```
DIM [Variable Name] As Data Type

DIM [Variable Name] AS STRING
DIM [Variable Name] AS INTEGER
DIM [Variable Name] AS LONG
DIM [Variable Name] AS SINGLE
DIM [Variable Name] AS DOUBLE
```

Examples:

```
DIM MyName AS STRING
DIM Num1 AS INTEGER
DIM Num2 AS SINGLE
DIM Answer AS SINGLE
```

Code:

```
DIM Num1 AS INTEGER
DIM Num2 AS LONG
DIM Num3 AS SINGLE
DIM Num4 AS DOUBLE
DIM Header AS STRING
CLS
Header = "This is another example"
Num1 = 5
Num2 = 56000
Num3 = 45.635
Num4 = 66000.5634#

PRINT Header
PRINT Num1 + Num2 + Num3 + Num4
```

Remember that selecting the right data type for the variable is very important to make the program run properly.

Topic: 2.2.1 Programming concepts

Using declared constants

Constants will be declared at the start of the main program. The following shows the declaration of constants for Pi and the Value added tax rate.

- CONST Pi = 3.14159
- CONST VatRate = 0.175

Declaring constants at the start of a program means that maintenance is made easier for two reasons:

- If the value of a constant changes, it only has to be changed in the one part of the program where it has been declared, rather than in each part of the program in which it is used;
- the code is easier to interpret:

For example:

- Total=VatRate*CostPrice
- Total=0.175*CostPrice

In the first statement, one can easily understand that the Total is the product of 2 variables

whereas; the VATRate was manually entered in the second statement creating ambiguity and a chance that the VAT could be entered incorrectly.

Understand and use basic data types

Define and use different data types e.g. integer, real, boolean, character and string

A data type is a method of interpreting a pattern of bits.

Data types and data structures

Intrinsic data types

Intrinsic data types are the data types that are defined within a particular programming language. There are numerous different data types. They are used to make the storage and processing of data easier and more efficient. Different databases and programming systems have their own set of intrinsic data types, but the main ones are:

- Integer;
- Real;
- Boolean;
- String;
- Character;
- Container

Topic: 2.2.1 Programming concepts

Integer
An integer is a positive or negative number that does not contain a fractional part. Integers are held in pure binary for processing and storage. Note that some programming languages differentiate between short and long integers (more bytes being used to store long integers).

Real
A real is a number that contains a decimal point. In many systems, real numbers are referred to as singles and doubles, depending upon the number of bytes in which they are stored.

Boolean
A Boolean is a data-type that can store one of only two values – usually these values are True or False. Booleans are stored in one byte – True being stored as 11111111 and False as 00000000.

String
A string is a series of alphanumeric characters enclosed in quotation marks. A string is sometimes just referred to as 'text'. Any type of alphabetic or numeric data can be stored as a string: "Birmingham City", "3/10/03" and "36.85" are all examples of strings. Each character within a string will be stored in one byte using its ASCII code; modern systems might store each character in two bytes using its Unicode. The maximum length of a string is limited only by the available memory.

Dates
In most computer systems dates are stored as a 'serial' number that equates to the number of seconds since January 1st, 1904 (thus they also contain the time). Although the serial numbers are used for processing purposes, the results are usually presented in one of several 'standard' date formats – for example, dd/mm/yyyy, or dd MonthName, yyyy. Dates usually take 8 bytes of storage.

Notes:
- if dates or numbers are stored as strings then they will not be sorted correctly; they will be sorted according to the ASCII codes of the characters – "23" will be placed before "9";
- Telephone numbers must be stored as strings or the leading zero will be lost.

Character
A character is any letter, number, punctuation mark or space, which takes up a single unit of storage (usually a byte).

Topic: 2.2.1 Programming concepts

Comparison of the common data types:

Data Type	Example value	Storage Required
Integer	42	4 bytes
Real	23.1	4 or 8 bytes
Boolean	True	1 byte
String	"Hello World"	1 byte for each character (if using Unicode, then 2 bytes are required for each character)
Character	"X"	1 byte

Program construct in Basic:

1.1) 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) will counts backwards.

1.2) Totaling

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*

```
| 10 x = 1                                     |
| 20 sum = 0                                   |
| 30 print x                                   |
| 40 input "enter a number";n                  |
| 50 sum = sum + n                             |
| 60 x = x + 1                                 |
| 70 if x < 11 then goto 30                    |
| 80 print "The sum of the numbers you gave is";sum |
```

Topic: 2.2.1 Programming concepts

Sequence: Execution of the program usually following the order in which the steps are written.

Selection: A decision between alternative routes through the program that involves at least one condition.

Repetition: A step or sequence of steps repeated within the program as an iteration or loop.

The structure of procedural programs

Statement, subroutine, procedure, function, parameter, loop

Procedural programs are ones in which instructions are executed in the order defined by the programmer.

Procedural languages are often referred to as third generation languages and include FORTRAN, ALGOL, COBOL, BASIC, and PASCAL.

Statement
A statement is a single instruction in a program, which can be converted into machine code and executed.

In most languages a statement is written on a single line, but some languages allow multiple lines for single statements.

Examples of statements are:

```
DIM name As String
A=X*X
While x < 10
```

Procedure
A procedure is a subroutine that performs a specific task without returning a value to the part of the program from which it was called.

Function
A function is a subroutine that performs a specific task and returns a value to the part of the program from which it was called.

Note that a function is ‘called’ by writing it on the right hand side of an assignment statement.

Topic: 2.2.1 Programming concepts

Parameter
A parameter is a value that is 'received' in a subroutine (procedure or function).

The subroutine uses the value of the parameter within its execution. The action of the subroutine will be different depending upon the parameters that it is passed.

Parameters are placed in parenthesis after the subroutine name. For example:
Square (5) 'passes the parameter 5 – returns 25
Square (8) 'passes the parameter 8 – returns 64

Square(x) 'passes the value of the variable x
Understand, create and use subroutines (procedures and functions), including the passing of parameters and the appropriate use of the return value of functions

Use subroutines to modularize the solution to a problem

Subroutine/sub-program
A subroutine is a self-contained section of program code which performs a specific task and is referenced by a name.
A subroutine resembles a standard program in that it will contain its own local variables, data types, labels and constant declarations.

There are two types of subroutine. These are procedures and functions.

- Procedures are subroutines that input, output or manipulate data in some way
- Functions are subroutines that return a value to the main program.

A subroutine is executed whenever its name is encountered in the executable part of the main program. The execution of a subroutine by referencing its name in the main program is termed 'calling' the subroutine.

The advantage of using procedures and functions are that:
- The same lines of code are re-used whenever they are needed – they do not have to be repeated in different sections of the program.
- A procedure or function can be tested/improved/rewritten independently of other procedures or functions.
- It is easy to share procedures and functions with other programs – they can be incorporated into library files which are then 'linked' to the main program;

- A programmer can create their own routines that can be called in the same way as any built-in command.

Topic: 2.2.1 Programming concepts

Sub Statement (Visual Basic)

Declares the name, parameters, and code that define a **Sub** procedure.

```
Sub name [ (Of typeparamlist) ] [ (parameterlist) ]
    [ statements ]
    [ Exit Sub ]
    [ statements ]
End Sub
```

All executable code must be inside a procedure. Use a **Sub** procedure when you don't want to return a value to the calling code. Use a **Function** procedure when you want to return a value.

Sample Code:

```
Sub computeArea(ByVal length As Double, ByVal width As Double)
    ' Declare local variable.
    Dim area As Double
    If length = 0 Or width = 0 Then
        ' If either argument = 0 then exit Sub immediately.
        Exit Sub
    End If
    ' Calculate area of rectangle.
    area = length * width
    ' Print area to Immediate window.
    Debug.WriteLine(area)
End Sub
```

Function Statement (Visual Basic)

Declares the name, parameters, and code that define a Function procedure.

```
Function name [ (Of typeparamlist) ] [ (parameterlist) ] [ As returntype ]
    [ statements ]
    [ Exit Function ]
    [ statements ]
End Function
```

Sample Code:

```
Function myFunction(ByVal j As Integer) As Double
    Return 3.87 * j
End Function
```

Topic: 2.2.1 Programming concepts

Convert following pseudocode into programs using any high level language:

The following five examples use the above pseudocode terms. These are the same problems discussed in section 3.1 using flow charts – both methods are acceptable ways of representing an algorithm.

2.1 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 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
```

For example,

```
count ← 0
while count < 5001
      input house
      if house > 50000 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)

Topic: 2.2.1 Programming concepts

2.2 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
```

2.3 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
```

Topic: 2.2.1 Programming concepts

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
```

2.4 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
```

Topic: 2.2.1 Programming concepts

2.5 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)

Topic: 2.2.1 Programming concepts

Questions:

May/June 2006 P1

(a) Give two benefits of using a high-level language for writing programs. [2]

(b) State one type of program that would be written in a low-level language rather than a high-level language and give a reason why. [2]

Oct/Nov 2007 P1

Give two differences between high level languages and low level languages.

Oct/Nov 2009 P1

Give two advantages of using high level languages when writing new computer software rather than using low level languages

May/June 2010 P11

10 (a) Compilers and interpreters translate high-level languages. Give two differences between compilers and interpreters.

(b) Programs can be written using high-level or low-level languages.

Give one advantage of using each method.

High-level language advantage

Low-level language advantage

(c) What is meant by top-down design when developing new software?

May/June 2011 P12

```
1 h = 0
2 c = 0
3 REPEAT
4 READ x
5 IF x > h THEN x = h
6 c = c + 1
7 PRINT h
8 UNTIL c < 20
```

The above code is an example of a high-level language.

Give TWO features of a high-level language. [2]

(c) The code is to be interpreted rather than compiled.

Give ONE difference between these two methods. [1]

Topic: 2.2.1 Programming concepts

May/June 2012 P12

Look at these two pieces of code:

A:

```
      CLC
      LDX #0
loop: LDA A,X
      ADC B,X
      STA C,X
      INX
      CPX #16
      BNE loop
```

B:

```
FOR Loop = 1 TO 4
    INPUT Number1, Number2
    Sum = Number1 + Number2
    PRINT Sum
NEXT
```

(a) Which of these pieces of code is written in a high-level language? [1]
(b) Give one benefit of writing code in a high-level language. [1]
(c) Give one benefit of writing code in a low-level language. [1]
(d) High-level languages can be compiled or interpreted.
Give two differences between a compiler and an interpreter. [2]