Topic: 2.3.1 Programming basics

Writing Programs

What are good program writing techniques?

Programmers should write code that is self-documenting and split into small sections.

Specifically, the programmers should:

- use meaningful identifier names for variables, constants and subroutines;
- use declared constants;
- annotate code with comments;
- indent code within iteration and selection;
- split the code into modules when possible;

The need for good program-writing techniques

It is important for a programmer to use good programming techniques when writing code so that the code:

- can be easier to understand by other programmers (who are either part of the programming team, or who will be responsible for the maintenance of the program in the future);
- can be understandable by the programmer himself in the future;
- is split into smaller blocks which are easier to debug individually and update;
- is less prone to errors – because the meaningful variable names and comments make it self-documenting and easier to read.

Topic: 2.3.1 Programming basics

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 during the program's execution.

Identifier

Identifier is the name given to a variable, constant or subroutine.

Assigning a variable or constant an identifier name means that it can be easily referenced from any part of the program within its scope and it helps to make the program more readable.

Reserved word/keyword

Reserved words (occasionally called keywords) are one type of grammatical construct in programming languages. These words have special meaning within the language and are predefined in the language's formal specifications. Typically, reserved words include labels for primitive data types in languages that support a type system, and identify programming constructs such as loops, blocks, conditionals, and branches.

Topic: 2.3.1 Programming basics

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
```

Intrinsic variable declarations

Some programming languages require intrinsic variable declarations. This means that variables must be declared before they are used.

The advantage of intrinsic variable declaration is that it cuts out possible errors due to the misspelling of a variable name – if a variable is used that has not been declared, the programming translator will identify it as an error.

```
DIM Number As Integer
Number= Nubmer+1 'This will be flagged as an Error!
```

There are two types of variables used by programmers and they are categorized according to their scope.

Topic: 2.3.1 Programming basics

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 coinciding.

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 VAT rate

```
CONST Pi = 3.142
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:

```
Total=VatRate*CostPrice ‘allows a greater understanding
```

rather than

```
Total=0.175*CostPrice
```

Topic: 2.3.1 Programming basics

Assignment

An assignment is an instruction in a program that places a value into a specified variable.

Some typical assignments are:

```
TheLength = 20.5
TheUsersName$ = “Charlie”
TheArea = TheWidth * TheLength
TotalCost = LabelledCost + 15
Counter = Counter + 1
```

Note that the last example is a common method used to increment the value of a variable. It could be read as:

“The new value of Counter is its existing value plus one"

Type Mismatch errors

A type Mismatch error occurs in a program when a variable has been declared as one data type, but it is later assigned a value that is of an incompatible data type.

The following code will produce a ‘Type Mismatch’ error because “Charlie" is not an integer:

```
DIM MyCounter AS Integer
MyCounter = “Charlie”
```

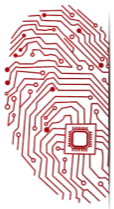
Other Type Mismatches will be produced by the following:

```
DIM RentalDateAs Date
MemberRentalDate = “September”
DIM ShoeSizeAs Integer
JohnsShoeSize = 10.3
```

Note that a variable that is declared as a string will never produce a type mismatch error.

Arithmetic Operator

Operator	Meaning/purpose	Example	Result
+	Addition	Result=3+5	8
–	Subtraction	Result=3–7	-4
*	Multiplication	Result=3*7	21



Topic: 2.3.1 Programming basics

Powers

Operator	Meaning/purpose	Example	Result
^	Power	`Result=3^2`	9

Division

A result of a division such as 17 ÷ 4 can be expressed either as a real (4.25) or as two integers (4 remainder 1).

The integer method, in most programming languages, uses the operators DIV and MOD.

Operator	Meaning/purpose	Example	Result
/	Division	`Result=14/5`	2.8
DIV	Integer Division – returns the result of a division after ignoring the decimal portion of all numbers	`14 DIV 5` `7.2 DIV 3.9 (= 7 DIV 3)`	2 2

MOD	Remainder – returns the remainder of the division of two integers	`14 MOD 5`
		`18 MOD 6`
		`3 MOD 7`
			4
			0
			3

Topic: 2.3.1 Programming basics

Relational operators (=, <, <=, >, >= and <>)

Relational operators are used in the format: [Expression] [Operator] [Expression] and will always return a Boolean (True or False) value.

Relational operators are typically used with the "IF" selection and also within conditional loops

(REPEAT-UNTIL or WHILE-WEND).

In the following table, the variables "a" and "name$" have the following assignments:

```
a=3+5
name$="JAMES"
```

Operator	Meaning/purpose	Example	Result
=	Equal – tests if two expressions are identical	IF a=8 IF name$="JANET" IF a=14	True False False
<	Less Than – tests if the first expression is less than the second	IF a<8 IF name$<"JANET" IF a<14	False True True
>	Greater Than – tests if the first expression is greater than the first	IF a>8 IF name$>"JANET" IF a>14	False False False
<>	Not Equal – tests if two expressions are different	IF a<>8 IF name$<>"JANET" IF name$<>"JAMES" IF a<>14	False True False True
<=	Less Than or Equal	IF a<=8 IF name$<="JANET" IF a<=14	True True True
>=	Greater Than or Equal	IF a>=8 IF name$>="JANET" IF a>=14	True False False

Topic: 2.3.1 Programming basics

Boolean operators AND, OR and NOT

AND and OR

The AND & OR operators always return a Boolean result and are used in the format:

```
[Boolean] [Operator] [Boolean]
```

The following 'truth' table summarizes the result of the Boolean operations:

Values Results

Values		Results	
X	Y	X AND Y	X OR Y
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

NOT

The NOT operator reverses the result of the Boolean expression and is used in the format:

```
NOT [Boolean]
```

The following truth table summarizes the NOT operation:

X	NOT X
True	False
False	True

Topic: 2.3.1 Programming basics

Examples of Boolean 'logic'

Consider the following algorithm, which is used to monitor a printer and display its output via a LCD display in the front panel:

```
IF NOT(PaperTrayEmpty) AND (FilesWaiting> 0) THEN
OUTPUT "PRINTING…"
ELSE
OUTPUT "PLEASE ADD PAPER"
END IF
```

If the values of the variables are:

```
PaperTrayEmpty = False
FilesWaiting = 2
```

Then the output will be "PRINTING…"

The following table shows why:

		X	Y	X AND Y
PaperTrayEmpty	FilesWaiting	NOT(PaperTrayEmpty)	FilesWaiting > 0	
False	2	True	True	True

Other options are shown in this table:

		X	Y	X AND Y
PaperTrayEmpty	FilesWaiting	NOT(PaperTrayEmpty)	FilesWaiting > 0	
True	2	False	False	False
True	0	False	False	False
False	0	True	False	False

in whichPaperTrayEmpty = False.

To avoid this incorrect message, the algorithm should be rewritten using a nested IF, as shown on the next page:

Topic: 2.3.1 Programming basics

```
IF PaperTrayEmpty THEN
  OUTPUT “PLEASE ADD PAPER”
ELSE
  IF FilesWaiting> 0 THEN
    OUTPUT “PRINTING…”
  ELSE
    OUTPUT “STATUS OK”
  END IF
END IF
```

Topic: 2.3.1 Programming basics

Meaningful identifier names

Identifiers are used to give names to constants and variables. They are also used to name procedures, functions, and the main program.

Naming conventions

Most of the identifier names must conform to the following rules (different programming languages may have slightly different rules):

- they must be unique;
- spaces must not be used;
- they must begin with a letter of the alphabet;
- the rest of the identifier must not contain punctuation – it may only consist of a mixture of letters and digits (A–Z, a–z and 0–9) and the underscore character '_';
- they must not be a 'reserved' word – eg Print, Repeat, For, Dim, Loop, etc.

Recommended naming policies

Do not use spaces within identifier names – even with programming languages where they are permitted. Instead, use the underscore character '_' or, better yet, type names in lowercase except the first letter of each word, which should be typed in uppercase.

Examples of good identifier names:

```
FirstName              LastName              PostCode
TelephoneNumber        WeightAtBirth         TestScore
AverageHeight
```

Further clarity can be given to identifier names by including a prefix that identifies the data type.

The above identifiers would be clearer if given the following prefix data types:

```
strFirstName           strLastName       strPostCode
strTelephoneNumber     sglWeightAtBirth        intTestScore
sglAverageHeight
```

Initializing a variable means setting it to a starter value. Initialization will usually set the value of an integer to 0 or 1 and a string to Empty ("").

Topic: 2.3.1 Programming basics

The following code initializes the variable Counter to zero before it is used in the iteration:

```
Counter=0 (the variable is initialized)
REPEAT
Counter=Counter+1
…
…
UNTIL Counter=10
```

Initializing a variable ensures that its value has not be been retained from a previous use of the routine and that the value has not been accidently set in another part of the program – this helps avoid errors.

Comments/remarks

Comments (originally called remarks) are added to program code to provide useful, or essential, documentation for the programmer and other people who read the code.

All programs/subroutines should contain comments to indicate:

- the details of the person who wrote the code;
- the date that the code was written;
- the purpose of the code;
- how parts of the code work;
- the use of certain variables.

Comments can be used anywhere in a program – either as separate lines by themselves or on the same line as executable code provided they appear to the right of any code that will execute.

Comments are usually indicated by the use of an apostrophe (‘), or two forward slashes (//).

```
PROCEDURE Arrange(NumberOfScores)
‘Procedure to move the largest element in an array to the end
‘Coded by J Jones, 02/03/09

DIM Ptr, Temp As Integer ‘ptr is the value of the array index

FOR Ptr = 1 TO NumberOfScores‘go through each element

  ‘test the relative values of the elements
  IF Scores(Ptr) < Scores(Ptr + 1) THEN
  ‘use a temporary store to help swap the elements

    Temp = Scores(Ptr)
    Scores(Ptr) = Scores(Ptr + 1)
    Scores(Ptr + 1) = Temp
  END IF
NEXT Ptr‘increment the index to examine the next element
END PROCEDURE
```

Topic: 2.3.1 Programming basics

Note that comments are ignored when the program code is compiled and so they are not present within the stand-alone application.

Indentation should be used within iteration and selection statements so that it is clear which instructions go together.

Examples:

Original code

```
INPUT Number
Total=0
WHILE Number > 0 THEN
Left=Number MOD 10
Right = Number DIV 10
Total=Total+Left
Number=Right
END WHILE
```

Indented code

```
INPUT Number
Total=0
WHILE Number > 0 THEN
  Left=Number MOD 10
  Right = Number DIV 10
  Total=Total+Left
  Number=Right
END WHILE
```

Original code

```
FUNCTION TEST(X)
IF X=1 THEN
PRINT 1
RETURN 1
ELSE
Number=X*Test(X-1)
PRINT Number
RETURN Number
END IF
END TEST
```

Indented code

```
FUNCTION TEST(X)
  IF X=1 THEN
    PRINT 1
    RETURN 1
  ELSE
    Number=X*Test(X-1)
    PRINT Number
    RETURN Number
  END IF
END TEST
```