3.2 Programming

3.2.1 Data types

Content	Additional information	Chk
Understand the concept of a **data type**.		
Understand and use the following appropriately: • integer • real (float) • Boolean • character • string.		

3.2.2 Programming concepts

Content	Additional information	Chk
Use, understand and know how the following statement types can be combined in programs: • **variable declaration** • **constant declaration** • **assignment** • **iteration** • **selection** • **subroutine** (procedure/function).	The three combining principles (sequence, iteration/repetition and selection/choice) are basic to all programming languages. Students should be able to write programs using these statement types. They should be able to interpret algorithms that include these statement types. Students should know why named constants and variables are used.	
Use definite and indefinite iteration, including indefinite iteration with the condition(s) at the start or the end of the iterative structure.	A theoretical understanding of condition(s) at either end of an iterative structure is required. Examples are: FOR; WHILE; REPEAT/UNTIL; DO/WHILE	
Use **nested selection** and **nested iteration** structures.		
Use meaningful **identifier names** and know why it is important to use them.	Identifier names include names for variables, constants and subroutine names.	

3.2.3 Arithmetic operations in a programming language

Content	Additional information	Chk
Be familiar with and be able to use: • addition • subtraction • multiplication • real division • integer division, including remainders.	Integer division, including remainders, uses modular arithmetic, eg: • Integer division: the integer quotient of 11 divided by 2 (11 DIV 2) = 5 • Remainder: the remainder when 11 is divided by 2 (11 MOD 2) = 1	

3.2.4 Relational operations in a programming language

Content	Additional information	Chk
Be familiar with and be able to use: • equal to • not equal to • less than • greater than • less than or equal to • greater than or equal to.	Students should be able to use these operators within their own programs and be able to interpret them when used within algorithms. In pseudo-code we use the symbols: =, ≠, <, >, ≤, ≥	

3.2.5 Boolean operations in a programming language

Content	Additional information	Chk
Be familiar with and be able to use: • NOT • AND • OR.	Students should be able to use these operators, and combinations of these operators, within conditions for iterative and selection structures.	

3.2.6 Data structures

Content	Additional information	Chk
Understand the concept of data structures.		
Use **arrays** (or equivalent) in the design of solutions to simple problems.	Only one and two-dimensional arrays are required.	
Use **records** (or equivalent) in the design of solutions to simple problems.		

3.2.7 Input/output

Content	Additional information	Chk
Be able to obtain **user input** from the keyboard.		
Be able to output data and information from a program to the computer display.		

3.2.8 String handling operations in a programming language

Content	Additional information	Chk
Understand and be able to use: • length • position • **substring** • **concatenation** • convert character to character code • convert character code to character • string conversion operations.	Expected string conversion operations: • string to integer • string to real • integer to string • real to string.	

3.2.9 Random number generation in a programming language

Content	Additional information	Chk
Be able to use random number generation.	An understanding of how pseudo-random numbers are generated is not required.	

3.2.10 Structured programming and subroutines (procedures and functions)

Content	Additional information	Chk
Understand the concept of **subroutines**.	Know that a subroutine is a named 'out of line' block of code that may be executed (called) by simply writing its name in a program statement.	
Explain the advantages of using subroutines in programs.		
Describe the use of **parameters** to pass data within programs.	Students should be able to use subroutines that require more than one parameter. Students should be able to describe how data is passed to a subroutine using parameters.	
Use subroutines that return values to the calling routine.	Students should be able to describe how data is passed out of a subroutine using return values.	
Know that subroutines may declare their own variables, called **local variables**, and that local variables usually • only exist while the subroutine is executing • are only accessible within the subroutine.		
Use local variables and explain why it is good practice to do so.		

Content	Additional information	Chk
Describe the **structured approach** to programming.	Students should be able to describe the structured approach including modularised programming, clear, well documented interfaces (local variables, parameters) and return values.	
Explain the advantages of the structured approach.		

3.2.11 Robust and secure programming

Content	Additional information	Chk
Be able to write simple **data validation** routines.	Students should be able to use data validation techniques to write simple routines that check the validity of data being entered by a user. The following validation checks are examples of simple data validation routines: • checking if an entered string has a minimum length • checking if a string is empty • checking if data entered lies within a given range (eg between 1 and 10).	
Be able to write simple **authentication routines**.	Students should be able to write a simple authentication routine that uses a username and password.	
Understand what is meant by **testing** in the context of algorithms and programs. Be able to correct errors within algorithms and programs.		
Understand what **test data** is and describe the following forms of test data: • **normal** (typical) • **boundary** (extreme) • **erroneous** data.	Example of boundary data: • if data is allowed in the range 1 to 10, boundary values are 0, 1, 9, 10 – i.e. either side of the allowed boundary	
Be able to select and justify the choice of suitable test data for a given problem.		
Understand that there are different types of error: • **syntax error** • **logic error**		
Be able to identify and categorise errors within algorithms and programs.		