ZNOTES.ORG

UPDATED TO 2023-2025 SYLLABUS

CAIE IGCSE

COMPUTER SCIENCE

SUMMARIZED NOTES ON THE THEORY SYLLABUS

Prepared for Nyasha for personal use only.

1. Algorithm Design & Problem-Solving

1.1. Program Development Life Cycle (PDLC)

- Analysis
- Design
- Coding
- Testing
- Maintenance

Analysis

- Before solving a problem, it is essential to define and document the problem clearly, known as the "requirements specification" for the program.
- The analysis stage involves using tools like abstraction and decomposition to identify the specific requirements for the program.
- Abstraction focuses on the essential elements needed for the solution while eliminating unnecessary details and information.
- Decomposition involves breaking down complex problems into smaller, more manageable parts that can be solved individually.
- Daily tasks can be decomposed into constituent parts for easier understanding and solving.

Design

- The program specification derived from the analysis stage is used as a guide for program development.
- During the design stage, the programmer should clearly understand the tasks to be completed, the methods for performing each task, and how the tasks will work together.
- Documentation methods such as structure charts, flowcharts, and pseudocode can be used to document the program's design formally.

Coding and iterative testing

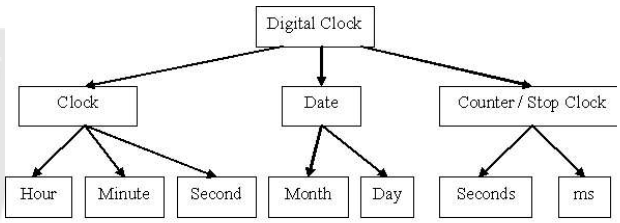
- The program or set of programs is developed based on the design.
- Each module of the program is written using a suitable programming language.
- Testing is conducted to ensure that each module functions correctly.
- Iterative testing is performed, which involves conducting modular tests, making code amendments if necessary, and repeating tests until the module meets the required functionality.

Testing

- The completed program or set of programs is executed multiple times using various test data sets.
- This testing process ensures that all the tasks within the program work together as specified in the program design.
- Running the program with different test data can identify and address potential issues and errors.
- The testing phase aims to verify the overall functionality and performance of the program by evaluating its behaviour with various inputs.

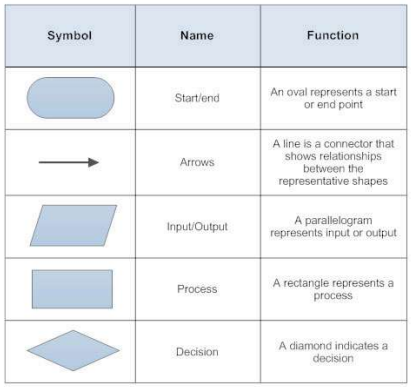
1.2. Structure Diagrams

- Every computer system is made up of sub-systems, which are in turn made up of further sub-systems.
- **Structure Diagrams** – The breaking down of a computer system into sub-systems, then breaking each sub-system into smaller sub-systems until each one only performs a single action. A structure diagram diagrammatically represents a top-down design. Example below.



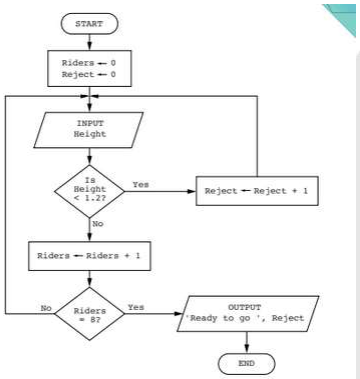
1.3. Pseudocode & Flowcharts

- **Pseudocode -** Verbal representation of an algorithm (a process or set of steps) and flowcharts are a diagrammatic representation.
- **Flowcharts:** A flowchart shows diagrammatically the steps required to complete a task and the order that they are to be performed
- **Algorithm:** These steps, together with the order, are called an algorithm



Symbol	Name	Function
	Start/end	An oval represents a start or end point
	Arrows	A line is a connector that shows relationships between the representative shapes
	Input/Output	A parallelogram represents input or output
	Process	A rectangle represents a process
	Decision	A diamond indicates a decision

An example of a flowchart is given below from a past paper question in which all of the functions of a flowchart are shown:



This flowchart's task is to check if a rider's height is more the requirement (1.2) in this case. It then counts until the accepted riders are 8. After they are 8, it outputs the number of rejected riders and tells the rest that they are ready to go!

2. Pseudocode

- Declaration & Usage of Variables & Constants
  - Variable – Store of data which changes during execution of the program (due to user input)
  - Constant – Store of data that remains the same during the execution of the program
- Basic Data Types
  - Integer – Whole Number e.g. 2; 8; 100
  - Real – Decimal Number e.g. 7.00; 5.64
  - Char – Single Character e.g. 'a'; 'Y'
  - String – Multiple Characters (Text) e.g. "ZNotes"; "COOL"
  - Boolean – Only 2 Values e.g. True/False; Yes/No; 0/1
- **Input & Output (READ & PRINT) –** Used to receive and display data to the user respectively. *(It is recommended to use input and output commands)*

```
INPUT Name
OUTPUT "Hello Mr." , Name

// Alternatively //

READ Name
PRINT "Hello Mr," , Name
```

- **Declaration of variable -** A variable/constant can be declared by the following manner

```
DECLARE [Variable Name] : [DATATYPE OF VARIABLE]
```

- **Array:** Array is similar to variable but it can store multiple values of same datatype under single name

```
DECLARE [ARRAYNAME] : ARRAY [Lower Limit : Upper Lim
```

- **Assignment -** Each variable is assigned using a left arrow.

```
[VARIABLE NAME] <---- [Value to be assigned]
ArrayName [IndexValue] <---- [Value to be assigned]
```

- Conditional Statements: IF...THEN...ELSE...ENDIF

```
IF [CONDITION] THEN
   [CONSEQUENCE]
ELSE
   [CONSEQUENCE]
ENDIF

IF GRADE > 100 THEN
   OUTPUT "INVALID"
ELSE
   OUTPUT "VALID"
ENDIF
```

CASE…OF…OTHERWISE…ENDCASE – Multiple conditions and corresponding consequences \n

```
CASE OF [VARIABLE]
   OPTION: [CONSEQUENCE]
OTHERWISE: [CONSEQUENCE]
ENDCASE

CASE OF GRADE
   GRADE>80: OUTPUT "A"
   GRADE>70: OUTPUT "B"
   GRADE>60: OUTPUT "C"
OTHERWISE: OUTPUT "FAIL"
ENDCASE
```

- **Loop Structures:**

```
FOR…TO…NEXT :  Will run for a determined/known
```

```
FOR [VARIABLE] ← [VALUE] TO [VALUE
   [CODE]
NEXT
```

REPEAT… UNTIL – Will run at least once till condition is satisfied; Verification is done after running code

```
REPEAT
   [CODE]
UNTIL [CONDITION]
```

WHILE…DO…ENDWHILE – May not ever run; Verification is done before running code

```
WHILE [CONDITION] DO
   [CODE]
ENDWHILE
```

Note: When using conditions in these loop structures and conditional statement, it has to be kept in mind that it can be done in two ways.

1. use of a Boolean variable that can have the value TRUE or FALSE
2. comparisons made by using coparison operators, where comparisons are made from left to right

```
IF [BOOLEAN VARIABLE]
  THEN
    OUTCOME
  ELSE
    OUTCOME
 ENDIF

IF ((CONDITION 1) OR ( CONDITION 2)) AND (CONDITION
 THEN
  OUTCOME
 ELSE
  OUTCOME
ENDIF
```

2.1.

Operator	Comparison
>	Greater than
<	Less than
=	Equal
>=	Greater than or equal
<=	Less than or equal
<>	Not equal
AND	Both
OR	Either
NOT	Not

2.2. Standard methods used in algorithm:

- Totalling :Totalling means keeping a total that values are added to

```
Total ← 0
FOR Counter ← 1 TO LoopLimit
 Total ← Total + ValueToBeTotalled
NEXT Counter
```

- Counting: Keeping a count of the number of times an action is performed is another standard method.

```
PassCount ← 0
FOR Counter ← 1 TO LoopLimit
 INPUT Value
 IF Value > Range
  THEN
    PassCount ← PassCount + 1
 ENDIF
NEXT Counter
```

- Maximum, minimum and average : Finding the largest and smallest values in a list are two standard methods that are frequently found in algorithms

```
MaxiumumValue <--- Array[1] MinimumValue <--- Array[
FOR Counter ← 2 TO LoopLimit
 IF Array[Counter] > MaximumValue
  THEN
   MaximumValue ← Array[Counter]
 ENDIF

 IF Array[Counter] < MinimumValue
  THEN
   MinimumValue ← Array[Counter]
 ENDIF
NEXT Counter

// Average//

Total ← 0
FOR Counter ← 1 TO NumberOfValues
 Total ← Total + StudentMark[Counter]
NEXT Counter
Average ← Total / NumberOfValues
```

- **Linear Search:** In a linear search, each item in the list is inspected sequentially until a match is found or the entire list is traversed.

```
INPUT Value
Found ← FALSE
Counter ← 0
REPEAT
 IF Value = Array[Counter]
 THEN
  Found ← TRUE
 ELSE
  Counter ← Counter + 1
 ENDIF
UNTIL Found OR Counter > NumberOfValues
IF Found
 THEN
  OUTPUT Value , " found at position " , Counter, "
 ELSE
  OUTPUT Value , " not found."
ENDIF
```

- **Bubble Sort:** Iteratively compare and swap adjacent elements in a list to sort them. Start from the first element and continue until the second-to-last element. After each pass, the last element is in its correct place. However, other elements may still be unsorted. Repeat the process, excluding the last element, until only one element remains or no swaps are needed.

```
First ← 1
Last ← 10
REPEAT
 Swap ← FALSE
 FOR Index ← First TO Last - 1
  IF Array[Index] > Array[Index + 1]
   THEN
    Temp ← Array[Index]
    Array[Index] ← Array[Index + 1]
    Array[Index + 1] ← Temp
    Swap ← TRUE
  ENDIF
 NEXT Index
 Last ← Last - 1
UNTIL (NOT Swap) OR Last = 1
```

2.3. Validation and Verification

To ensure the acceptance of reasonable and accurate data inputs, computer systems must thoroughly examine each data item before accepting it, and this is where Validation and Verification come into play!

Validation

Validation in computer systems involves automated checks to ensure the reasonableness of data before accepting it. If the data is invalid, the system should provide an explanatory message for rejection and allow another chance to enter the data.

There are multiple types of validation. These include:

Range check

A range check verifies that a numerical value falls within specified upper and lower limits.

```
REPEAT
 INPUT Value
 IF Value < MinimumValue OR Value > MaximumValue
  THEN
  OUTPUT "The student's mark should be in the range"
 ENDIF
UNTIL Value >= MinimumValue AND Value <= MaximumValu
```

Length check

This can either ensure that data consists of a precise number of characters.

```
OUTPUT "Please enter your value of ", Limit , "  cha
REPEAT
 INPUT Value
 IF LENGTH(Value) <> Limit
  THEN
  OUTPUT "Your value must be exactly" , Limit ," cha
 ENDIF
UNTIL LENGTH(Value) = Limit
```

It can also check if the data entered is a reasonable number of characters or not

```
OUTPUT "Please enter your value "
REPEAT
 INPUT Value
 IF LENGTH(Value) > UpperLimit OR LENGTH(Value) < Lo
  THEN
   OUTPUT "Too short or too long, please re-enter "
 ENDIF
UNTIL LENGTH(Value) <= UpperLimit AND LENGTH(Value)
```

Type check

A type check verifies that the entered data corresponds to a specific data type.

```
OUTPUT "Enter the value "
REPEAT
 INPUT Value
 IF Value <> DIV(Value, 1)
  THEN
   OUTPUT "This must be a whole number, please re-er
 ENDIF
UNTIL Value = DIV(Value, 1)
```

Presence check

A presence check checks to ensure that some data has been entered and the value has not been left blank

```
OUTPUT "Please enter the value "
REPEAT
 INPUT Value
 IF Value = ""
  THEN
   OUTPUT "*=Required "
 ENDIF
UNTIL Value <> ""
```

Format Check

A format check checks that the characters entered conform to a pre-defined pattern.

Check Digit

- A check digit is the final digit included in a code; it is calculated from all the other digits.
- Check digits are used for barcodes, product codes, International Standard Book Numbers (ISBN), and Vehicle Identification Numbers (VIN).

Verification

Verification is checking that data has been accurately copied from one source to another
There are 2 methods to verify data during entry (there are other methods during data transfer, but they are in paper 1)

1. Double Entry

- Data is inputted twice, potentially by different operators.
- The computer system compares both entries and if they differ, an error message is displayed, prompting the data to be reentered.

2. Screen/Visual check

- A screen/visual check involves the user manually reviewing the entered data.
- After data entry, the system displays the data on the screen and prompts the user to confirm its accuracy before proceeding.
- The user can compare the displayed data against a paper document used as an input form or rely on their own knowledge to verify correctness.

3. Test Data

- Test data refers to input values used to evaluate and assess the functionality and performance of a computer program or system.
- It helps identify errors and assess how the program handles different scenarios

3.1. Normal Data

- Normal data is the test data which accepts values in acceptible range of values of the program
- Normal data should be used to work through the solution to find the actual result(s) and see if they are the same as the expected result(s)
- e.g. in a program where only whole number values ranging from 0 to 100 (inclusive) are accepted, normal test data will be : 23, 54, 64 , 2 and 100

3.2. Abnormal Data

- Test data that would be rejected by the solution as not suitable, if the solution is working properly is called abnormal test data / erroneous test data.
- e.g. in a program where only whole number values ranging from 0 to 100 (inclusive) are accepted, abnormal data will be: -1, 151, 200, 67.2, “Sixty-Two” and -520

3.3. Extreme Data

- Extreme data are the largest and smallest values that normal data can take
- e.g. in a program where only whole number values ranging from 0 to 100 (inclusive) are accepted, extreme data will be: 0 and 100

3.4. Boundary Data

- This is used to establish where the largest and smallest values occur
- At each boundary two values are required: one value is accepted and the other value is rejected.
- e.g. in a program where only whole number values ranging from 0 to 100 (inclusive) are accepted, one example of boundary data will be: 100 and 101. 100 will be accepted and 101 will not be accepted

4. Trace Table

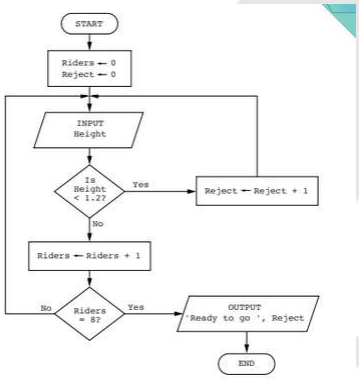
- A trace table is utilized to document the outcomes of every step in an algorithm. It is employed to record the variable's value each time it undergoes a change.
- A **dry run** refers to the manual process of systematically executing an algorithm by following each step in sequence.
- A trace table is set up with a column for each variable and a column for any output e.g.

A	B	C	X	OUTPUT
0	0	100		

Test data is employed to execute a dry run of the flowchart and document the outcomes in a trace table. During the dry run:

- Whenever a variable's value changes, the new value is recorded in the respective column of the trace table.
- Each time a value is outputted, it is displayed in the output column.

An example of trace table is given below using a past paper question:
Q: The flowchart below inputs the height of children who want to ride on a rollercoaster. Children under 1.2 metres are rejected. The ride starts when eight children have been accepted.



Complete the trace table for the input data: 1.4, 1.3, 1.1, 1.3, 1.0, 1.5, 1.2, 1.3, 1.4, 1.3, 0.9, 1.5, 1.6, 1.0

Riders	Reject	Height	OUTPUT
0	0		
1		1.4	
2		1.3	
	1	1.1	
3		1.3	
	2	1.0	
4		1.5	
	3	1.2	
5		1.3	
6		1.4	
7		1.3	
	4	0.9	
8		1.5	Ready to go 4

4.1. Identifying errors:

- Trace tables can be used to trace errors in a program. For example, if the requirement for the previous question would be to accept riders that are of height 1.2 too, rather than rejecting them, then the error would have been caught in the trace table as when 1.2 is entered, it would increment rejected which it shouldn't in our example

5. How to write an algorithm?

The ability to write an algorithm is very important for this syllabus and paper. Some key steps/points to be known in-order to write the perfect algorithm are as follows:

1. Make sure that the problem is clearly understood which includes knowing the purpose of the algorithm and the tasks to be completed by the algorithm.
2. Break the problem into smaller problems (e.g. in a program which outputs average values, divide the problem into multiple ones i.e. how to count the number of iterations and how to count the total of all values)
3. Identify the data that is needed to be saved into variables/constants/arrays and what datatype it is, and declare all the variables/constants/arrays accordingly, with meaningfull names
4. Decide on how you are going to construct your algorithm, either using a flowchart or pseudocode. If you are told how to construct your algorithm, then follow the guidance.
5. Construct your algorithm, making sure that it can be easily read and understood by someone else. Take particular care with syntax e.g. when conditions are used for loops and selection.
6. Use several sets of test data (Normal, Abnormal and Boundary) to dry run your algorithm and check if the expected results are achieved (a trace table can be used for this purpose) . If error is found, find the point of error in the trace table and fix it in the code.

Note: The algorithms that you have looked at so far in these notes were not designed with readability in mind because you needed to work out what the problem being solved was.

6. Programming

6.1. Programming Languages

There are many high-level programming languages to choose from. We will only be treating Python, Visual Basic, or Java.

- **Python** is an open-source, versatile programming language that encourages quick program development and emphasises code readability. The integrated development environment (IDE) showcased in this chapter is referred to as IDLE.
- **Visual Basic** is a popular programming language that is extensively used for Windows development. The integrated development environment (IDE) featured in this chapter is known as Visual Studio, which is utilised for capturing screenshots.
- **Java** is a widely adopted programming language utilised by numerous developers. The integrated development environment (IDE) employed for capturing screenshots in this chapter is known as BlueJ.

6.2. Programming Concepts

Constructs of a Program

- Data use – variables, constants and arrays
- Sequence – order of steps in a task
- Selection – choosing a path through a program
- Iteration – repetition of a sequence of steps in a program
- Operators use arithmetic for calculations and logic and Boolean for decisions.

Variables and Constants

- A **variable** within a computer program refers to a named storage unit with a value that can be modified throughout the program's execution. To enhance comprehension for others, it is advisable to assign significant names to variables.
- A **constant** within a computer program represents a named storage unit that holds a value which remains unchanged throughout the program's execution. Similar to variables, it is recommended to assign meaningful names to constants to enhance comprehensibility for others.

Data Types

- Different data types are assigned to computer systems for effective processing and storage.
- Data types allow data, such as numbers or characters, to be stored appropriately.
- Data types enable effective manipulation using mathematical operators for numbers and character concatenation.
- Some data types provide automatic validation.
- The types of datatypes are told in Chapter 1 already!

Input and Output

- Programs require input and output statements to handle data.
- In IGCSE Computer Science, algorithms and programs are designed to take input from a keyboard and output to a screen.
- Prompting the user with clear instructions for input is necessary for the user to understand what is expected.
- Input data in programming languages must match the required data type of the variable where it will be stored.
- By default, inputs are treated as strings, but commands can convert input to integer or real number data types.
- Users should be provided with information about the output/results for a program to be useful.
- Each output should be accompanied by a message explaining the result's meaning or significance.
- If an output statement has multiple parts, they can be separated by a separator character.

6.3. Basic Concepts

When writing the steps required to solve a problem, the following concepts need to be used and understood:

- Sequence
- Selection
- Iteration
- Counting and totalling
- String handling
- Use of operators.

Sequence

The ordering of the steps in an algorithm is very important. An incorrect order can lead to incorrect results and/or extra steps that are not required by the task.

Selection

Selection is a very useful technique, allowing different routes through the steps of a program. The code of this is explained in the notes of previous chapters.

Iteration

As explained in the previous chapter, we already

Totalling and Counting

As explained in the previous chapter, we already

String Handling

- Strings are used to store text and can contain various characters.
- An empty string has no characters, while the programming language specifies the maximum number of characters allowed.
- Characters in a string can be identified by their position number, starting from either zero or one, depending on the programming language.
- String handling is an important aspect of programming.
- In IGCSE Computer Science, you will need to write algorithms and programs for the following string methods:
  - Length: Determines the number of characters in a string, including spaces.
  - Substring: Extracts a portion of a string.
  - Upper: Converts all letters in a string to uppercase.
  - Lower: Converts all letters in a string to lowercase.
- These string manipulation methods are commonly provided in programming languages through library routines.

Finding the length of a string:

```
LENGTH("Text Here")

LENGTH(Variable)
```

Extracting a substring from a string:

```
SUBSTRING("Computer Science", 10, 7)
// returns the next 7 values starting from the 10th
SUBSTRING(Variable, Position, Length)
```

Converting a string to upper case

```
UCASE("Text here")

UCASE(Variable)
```

Converting a string to lowercase

```
LCASE("Text Here")

LCASE(Variable)
```

Arithmetic, Logical and Boolean Operators

As explained in the previous chapter, we already

Use of Nested Statements

- Selection and iteration statements can be nested, meaning one statement can be placed inside another.
- Nested statements help reduce code duplication and simplify testing of programs.
- Different types of constructs can be nested within each other, such as selection statements within condition-controlled loops or loops within other loops.

Procedures and Functions

Procedures and functions are defined at the start of the code.

- A **procedure** refers to a collection of programming statements organized under a single name, invoked at any given point in a program to execute a specific task.
- A function is a compilation of programming statements consolidated under a singular name, invoked at any moment within a program to accomplish a particular task. Unlike a procedure, a function also has the capability to return a value back to the main program.
- Parameters refer to variables that store the values of arguments passed to a procedure or function. While not all procedures and functions require parameters, some utilize them to facilitate their operations.

Procedures without parameters:

```
PROCEDURE ProcedureName ()
 [Commands]
ENDPROCEDURE
//Calling/running the procedure
CALL ProcedureName()
```

The procedure with parameters:

```
PROCEDURE ProcedureName (ParameterName : ParameterDa
 [Commands]
ENDPROCEDURE
//Calling/running the procedure
CALL ProcedureName (ParameterValue)
```

Function:

```
FUNCTION FunctionName (ParameterName : ParameterData
 [Commands]
 RETURN ValueToBeReturned
ENDFUNCTION
```

- When defining procedures and functions, the **header** is the first statement in the definition.
- The header includes:
  - The name of the procedure or function.
  - Parameters passed to the procedure or function, along with their data types.
  - The data type of the return value for a function.
- Procedure calls are standalone statements.
- Function calls are made as part of an expression, typically on the right-hand side.

Local and Global Variable

- Any part of a program can use a global variable – its scope covers the whole program
- A local variable can only be used by the part of the program it is declared in – its scope is restricted to that part of the program.

Note: Any variables/arrays made in this procedure and functions will be local and cannot be used out of these. To be made available all over the program, they must be declared globally in the following way.

```
DECLARE [VariableName] : DataType AS GLOBAL
```

6.4. Library Routines

- Programming language development systems often provide library routines that can be readily incorporated into programs.
- Library routines are pre-tested and ready for use, making programming tasks easier.
- Integrated Development Environments (IDEs) typically include a standard library of functions and procedures.
- Standard library routines perform various tasks, including string handling.
- MOD – returns the remainder of a division
- DIV – returns the quotient (i.e. the whole number part) of a division
- ROUND – returns a value rounded to a given number of decimal places
- RANDOM – returns a random number.

Examples:

- Value1 <--- MOD(10,3) returns the remainder of 10 divided by 3
- Value2 <---- DIV(10,3) returns the quotient of 10 divided by 3
- Value3 <--- ROUND(6.97354, 2) returns the value rounded to 2 decimal places
- Value4 <--- RANDOM() returns a random number between 0 and 1 inclusive

6.5. Creating a Maintainable Program

A maintainable program should:

- always use meaningful identifier names for variables, constants, arrays, procedures and functions
- be divided into modules for each task using procedures and functions
- be fully commented using your programming language's commenting feature

Commenting in pseudocode:

```
// Now the text written is commented and thus ignore

""
This method can also be used to comment
multiple lines but the singular line method
is more widely accepted and reccomended too
""
```

6.6. Arrays

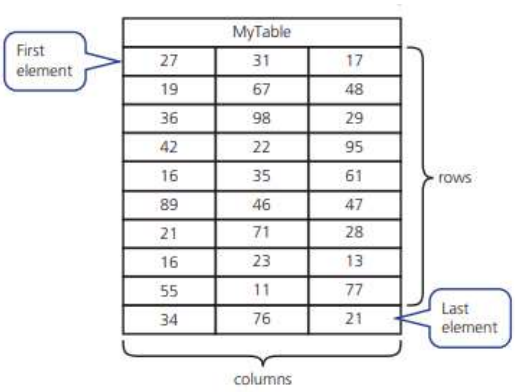
- An array is a data structure containing several elements of the same data type; these elements can be accessed using the same identifier name.
- The position of each element in an array is identified using the array's index.
- There are two types of arrays

One-Dimensional Array

Explained in the previous chapter in detail

Two-Dimensional Array

- A two-dimensional array can be referred to as a table with rows and columns.



- When a two-dimensional array is declared in pseudocode, the first and last index values for rows and the first and last index values for columns alongside the data type are included.

Declaring a 2D Array:

```
DECLARE Name : ARRAY[RowLower:RowUpper,ColumnLower:C
```

Filling a 2-D array using a loop:

```
FOR ColumnCounter ← 1 TO 3
 FOR RowCounter ← 1 TO 10
 OUTPUT "Enter next value "
 INPUT ArrayName [RowCounter, ColumnCounter]
 NEXT RowCounter
NEXT ColumnCounter
```

6.7. File Handling

- **Computer programs store data that will be needed again in a file.**

- Data stored in RAM is volatile and will be lost when the computer is powered off.
- Data saved to a file is stored permanently, allowing it to be accessed by the same program at a later date or by other programs.
- Stored data in a file can be transferred and used on other computers.
- The storage of data in files is a commonly used feature in programming.

Key point: When writing in a file, the program is outputing the data to the file, and when reading a file, the program in inputing the data from the file

\n There are 3 ways a file can be opened in a program i.e. to write, to read and to append

6.8. Writing in a file

```
OPENFILE "filename.txt" FOR WRITE

//When opening a file to write, all the data already


WRITEFILE "filename.txt" , Value

// The next command of WRITEFILE would be writen on

CLOSEFILE "filename.txt"
```

6.9. Reading a file:

```
OPENFILE "filename.txt" FOR READ
READFILE "filename.txt" , Variable
// The value in the line (which is identified by the
CLOSEFILE "filename.txt"
```

6.10. Reading a file till EOF:

```
OPENFILE "filename.txt" FOR READ
DECLARE DataVariable : STRING
WHILE NOT EOF("filename.txt) DO
 READFILE "filename.txt", DataVariable
 // here the line can be outputted or stored in an a
 //before the file ends has been read
ENDWHILE
```

7. Databases

A database is a well-organized compilation of data that enables individuals to retrieve information according to their specific requirements. The data contained within a database can encompass various forms such as text, numerical values, images, or any other type of digital content that can be stored on a computer system.

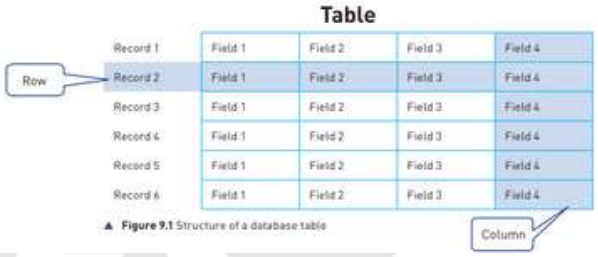
7.1. Why do we need a database?

- To store data about people, things, and events.
- Any modifications or additions need to be made only once, ensuring data consistency.
- All users access and utilize the same set of data, promoting uniformity.
- Relational databases store data in a non-repetitive manner, eliminating duplication.

7.2. What makes a database?

- Data is stored in **tables** in databases. Each table consists of a specific type of data e.g. cars. These tables HAVE to be named according to what they contain e.g. a table containing patient information will be ***PATIENT***
- These tables consist of **records** (rows). Each record consists of data about a single entity (a single item, person or event) e.g. a single car
- These tables also have columns that are knows an **fields**. These consist of specific information regarding the entities that are written later in records e.g. car name, car manufacturer etc.

Note: In this chapter, skills of dealing with a database are also required so working with Microsoft Access is needed to understand this chapter better. You have to be able to define a single-table database from given data storage requirements, choose a suitable primary key for a database table and also be able to read, complete and understand SQL scripts.



▲ **Figure 9.1** Structure of a database table

Source: Cambridge IGCSE and O Level Computer Science by Hodder Education

7.3. Validation in databases

- Database management software automatically provides some validation checks, while others need to be set up by the developer during construction.
- For example; The software automatically validates fields like "DateOfAdmission" in the ***PATIENT*** table to ensure data input is a valid date. \n

7.4. Basic Data Types

Each field will require a data type to be selected. A data type classifies how the data is stored, displayed and the operations that can be performed on the stored value. The datatypes for database are quite similar to original datatypes, however, there are a few differences.

Syllabus data type	Description	Access data type
text/alphanumeric	A number of characters	short text/long text
character	A single character	short text with a field size of or
Boolean	One of two values: either True or False, 1 or 0, Yes or No	Yes/No
integer	Whole number	number formatted as fixed wit zero decimal places
real	A decimal number	number formatted as decimal
date/time	Date and/or time	Date/Time

Note: Access datatype refers to the software Microsoft Access which is a DBMS (DataBase Management System). Here, databases could be worked upon in practical form

7.5. Primary Key

- Each record in a table represents a unique item, person, or event.
- To ensure reliable identification of these items, a field called the primary key is necessary.
- The primary key is a unique field that distinguishes each item within the data.
- In order to serve as a primary key, a field must have values that are never repeated within the table.
- An existing field can serve as a primary key if it is unique, such as the ISBN in the book table.
- In cases where all existing fields may contain repeated data, an additional field, such as "HospitalNumber," can be added to each record to serve as the primary key.

7.6. Structured Query Language - SQL

- Structured Query Language (SQL) is the standard language for writing scripts to retrieve valuable information from databases.
- By using SQL, we can learn how to retrieve and display specific information needed from a database.
- For instance, someone visiting a patient may only require the ward number and bed number to locate them in the hospital, while a consultant may need a list of the names of all the patients under their care. This can be done using SQL

SQL Scripts

- An SQL script is a collection of SQL commands that are used to perform a specific task, often stored in a file for reusability.
- To comprehend SQL and interpret the output of an SQL script, practical experience in writing SQL scripts is necessary.

Select Statements:

```
SELECT (fieldsname)
FROM (tablesname)
WHERE (condition)
ORDER BY (sortingcondition) ;
```

Selecting Sum of values in a table:

```
SELECT SUM ( fieldsname )
FROM (tablesname)
WHERE (condition)
ORDER BY (sortingcondition) ;
```

Counting the number of records where the field matches a specified condition

```
SELECT COUNT ( fieldsname )
FROM (tablesname)
WHERE (condition)
ORDER BY (sortingcondition) ;
```

==ORDER BY ASCENDING - sorts in ascending order.==
==ORDER BY DESENDING - sorts in descending order.==
Note: ORDER BY...is not necessary to add. It has to be only added if required!

Conditions often include values from fields, these values need to be stated in a form that matches the data type for the field.

Field type	Example value	General notes	Access notes
text	`'Mr Smith'`	Text field values should be in enclosed in single quotation marks.	Double quotation marks can also be used.
character	`'M'`	Character field values should be in enclosed in single quotation marks.	Double quotation marks can also be used.
Boolean	`TRUE`	Boolean can be `TRUE` or `FALSE`	Data type is Yes/No
integer	`12`	Integer field values should be whole numbers.	Allows integer or decimal values.
real	`12.01`	Real field values should be decimal numbers.	Allows integer or decimal values.
Date/time	`'22/11/2022'`	Date/time field values should be in enclosed in single quotation marks.	Date/time field values **must** be in enclosed in hashes (#).

7.7. Operators

Just like pseudocode, the operators used there can also be used here for conditions, however, a few more are also used in databases

Operator	Description
=	equal to
>	greater than
<	less than
>=	greater than or equal to
<=	less than equal to
<>	not equal to
BETWEEN	between a range of two values
LIKE	search for a pattern
IN	specify multiple values
AND	specify multiple conditions that must all be true
OR	specify multiple conditions where one or more conditions must be true
NOT	specify a condition that must be false

8. Boolean Logic

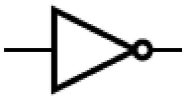
8.1. Logic Gates and their functions

Six types of logic gates

- NOT Gate
- AND Gate
- OR Gate
- NAND Gate
- NOR Gate
- XOR Gate

NOT gate: an inverter, $\overline{A}$

A	Output
0	1
1	0



AND gate: $A.B$

A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

OR gate: $A + B$

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

NAND gate: $\overline{A.B}$

A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

NOR gate: $\overline{A + B}$

A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0

XOR gate: A ⊕ B

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

9. Writing Logic Statements

Logic Statements is a way of showing all the logics that are in place for a logic circuit.

9.1. Writing from a logic circuit

1. Look at the ciruit and go around the logic gates used in the circuit
2. Go from the one output that is being given towards the input
3. Write the last gate (the first gate you walk through) in the middle and then, for each of the value coming into the gate, leave space at the side
4. If the value coming into the gate is coming from another gate, use a bracket for the gate's logic
5. Repeat process 3-4 till you are able to reach the input values fully

9.2. Writing from a truth table

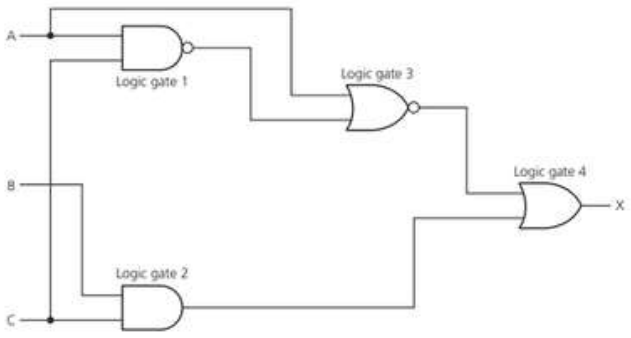
1. Create logic circuit fom the truth table (shown later)
2. Write the logic statement using the ciruit

9.3. Writing from a Problem statement

1. See what logics go in place in the statement to take place
2. Go from the logic of any 2 inputs at the start, and then keep on going until you are able to reach the final gate which gives the output
3. When writing the statement, make sure you show the logic statement where the output is 1

9.4. Example of a LOGIC STATEMENT

(B AND C) OR (A NOR (A NAND C)) is the logic statement for the following Logic Circuit



10. Creating Truth Tables

10.1. From Logic Circuits

1. Create a truth table with each input possible, creating every possible combination of inputs . **Tip: For the first input, write it in the combination of 1,0,1,0 and so on. For the second, go 1,1,0,0 and so on, and for the third one, go 1,1,1,1,0,0,0,0 going by the powers of 2 for each input. This would guarantee each possible combination**
2. Run through the circuit with the inputs and get the output that will be reached and write it accordingly

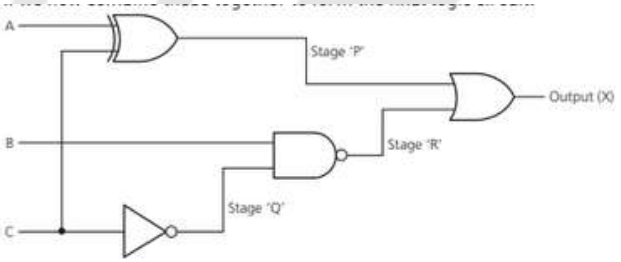
For logic statements, and problem statements, convert them to logic circuits first and then do the rest

10.2. Example

This is the example of a truth table of a logic circuit

Input values			Values at stages:			Output
A	B	C	'P'	'Q'	'R'	X
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	0	1	0	0
0	1	1	1	0	1	1
1	0	0	1	1	1	1
1	0	1	0	0	1	1
1	1	0	1	1	0	1
1	1	1	0	0	1	1

The circuit:



11. Logic Statements from Truth Tables

Input values			Output
A	B	C	X
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

1. Given the truth table above, take the rows where the output (x) is 1 (Rows 1, 2, 4, 5, 6, 7)
2. Create a logic expression from these rows (example, row 1 will be (NOT A AND NOT B AND NOT C) = X
3. Create logic expressions for all the rows with output 1 and connect them with OR gate

12. Exam-Style Question

A wind turbine has a safety system which uses three inputs to a logic circuit. A certain combination of conditions results in an output, X, from the logic circuit being equal to 1. When the value of X = 1 then the wind turbine is shut down.

The following table shows which parameters are being monitored and form the three inputs to the logic circuit.

▼ Table 10.20

parameter description	parameter	binary value	description of condition
turbine speed	S	0	turbine speed <= 1000 rpm
		1	turbine speed > 1000 rpm
bearing temperature	T	0	bearing temperature <= 80°C
		1	bearing temperature > 80°C
wind velocity	W	0	wind velocity <= 120 kph
		1	wind velocity >120 kph

The output, X, will have a value of 1 if any of the following combination of conditions occur:

Either	turbine speed <= 1000 rpm and bearing temperature > 80°C
Or	turbine speed > 1000 rpm and wind velocity > 120 kph
Or	bearing temperature <= 80°C and wind velocity > 120 kph

Design the logic circuit and complete the truth table to produce a value of X = 1 when either of the three conditions above occur.

1. The Conditions are given so make logic statements using the conditions and the table. (NOT S AND T) OR (S AND W) OR (NOT T AND W)
2. Make the logic circuit from the given equation
3. Make the truth table

CAIE IGCSE

Computer Science