

MOJZA

O Levels & IGCSE

COMPUTER SCIENCE NOTES

Paper 2: Programming and Algorithm

2210 & 0478

BY TEAM MOJZA

All rights reserved
www.mojza.org

CONTENTS

Pg 02 Unit 7 Algorithm Design and
Problem-Solving

Pg 11 Unit 8 Pseudocode

Pg 26 Unit 9 Databases

Pg 31 Unit 10 Boolean Logic

Unit 7: Algorithm design and Problem-solving

Program development life cycle

→ Consists of four main stages

- Analysis

→ Abstraction: Removal of unnecessary details and identification of the key elements of the problem

→ Decomposition: Breaking down a complex problem into smaller, easier-to-solve parts

→ Identification of the problem and requirements

- Design

→ Pseudocode

→ Flowcharts

→ Structure diagrams

→ How the problem is to be solved, what tasks are required for it and how they work with each other, and the order of those tasks is found out in this stage

- Coding

→ Program code is written using a programming language

→ Iterative testing of the separate modules of the program is carried out, ensuring they work as they are meant to

→ Most amendments to the code are carried out in this stage

- Testing

→ The completed program is tested with test data to spot any errors

Computer systems

- Made up of subsystems, which are made up of further sub-systems, and so on
- Made up of software, data, hardware, communications, and people
- The division can shown using top-down design
- Top-down design is the breaking down of a system into smaller sub-systems, and those subsystems into further sub-systems, until each sub-system performs a single action

Decomposition of a problem

- A problem can be decomposed by the identification of its component parts
- These include the inputs, processes, outputs, and storage required for the solution

- Inputs

- The data that is to be entered into the system for processing while it is active

- Processes

- The tasks that need to be performed on the data input and and/or the data previously stored in the system

- Outputs

- The data that needs to be displayed or printed for the users of the system
- The results of the various processes carried out in the system using the input data and/or the previously stored data

- Storage

- Data that needs to be stored in an appropriate medium in the system so that they can be used throughout the program as needed

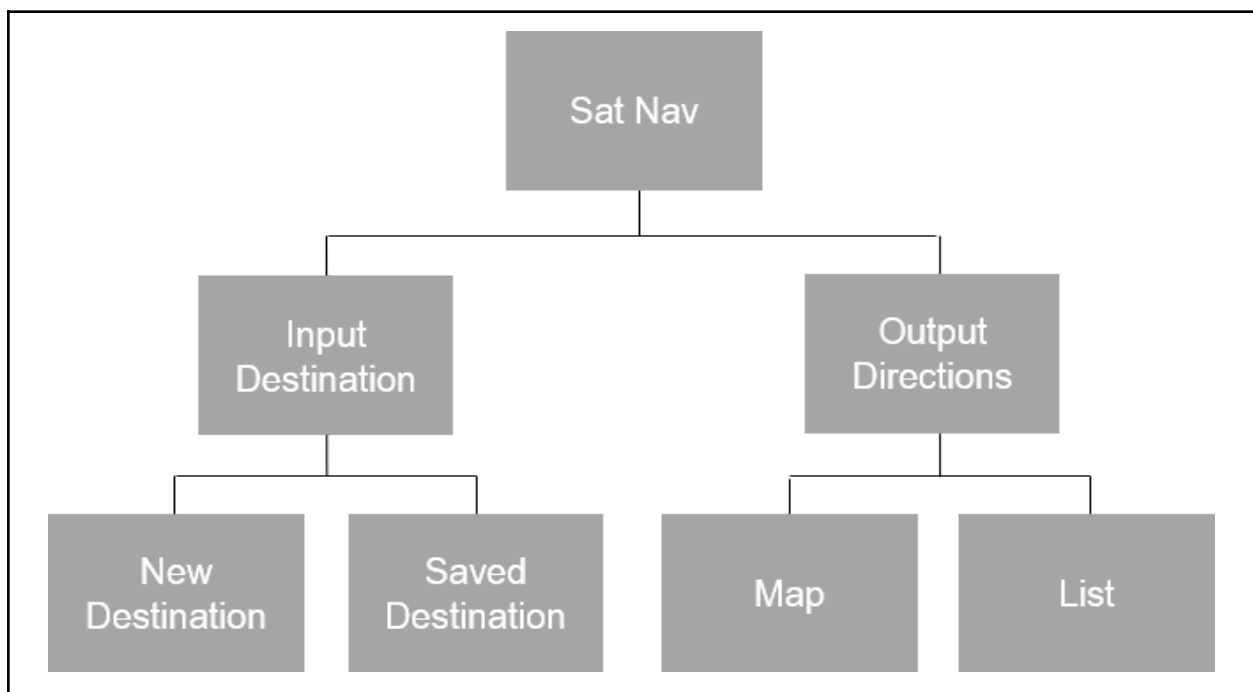
Methods used to design and construct solutions to a problem

- Solutions to problems need to be designed using thorough methods, to ensure that their purpose is clearly understood
- Include structure diagram, flowcharts, and pseudocode

- Structure diagrams





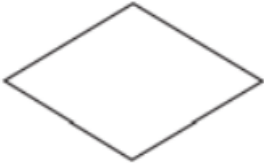

- Hierarchical diagrams that show the decomposition of a system
- Display top-down design diagrammatically
- Show the solution to a problem in a hierarchical way, dividing it into subsystems and dividing those subsystems into further sub-systems, until each sub-system performs a single action

- The following is a structure diagram of a part of a satellite navigation system that:
 - ↳ allows the user to enter details for a new destination or select a previously saved destination
 - ↳ displays directions in the form of a visual map or as a list



- Flowcharts

- Diagrammatically show the steps required, and the order in which they are to be carried out to complete a task
- The steps, along with the order, are called an algorithm
- Have standard flowchart symbols for different processes

Symbol	Name	Description
	Flow line	Represents control passing between connected shapes
	Process	Represents something being performed or done
	Subroutine	Represents a subroutine call that relates to another non-linked flowchart
	Input/Output	Represents data being input/output into or out of a flowchart
	Decision	Represents a decision (Yes/No or True/False) that results in two different possible outcomes
	Terminator	Represents the 'Start' or 'Stop' of a program

Standard methods of solution

- Linear search
- Bubble sort
- Totalling
- Counting
- Finding maximum, minimum and average values

- Linear search

- Used to search for a certain value/variable, etc.
- Checks all the elements of the data structure being searched

Example

```
OUTPUT "Please enter the name to find"
INPUT Name
Found ← FALSE
Counter ← 1
REPEAT
    IF Name = StudentName[Counter]
        THEN
            Found ← TRUE
        ELSE
            Counter ← Counter + 1
        ENDIF
UNTIL Found OR Counter > ClassSize
IF Found
    THEN
        OUTPUT Name, " found at position ", Counter, " in the list"
    ELSE
        OUTPUT "Name not found"
ENDIF
//pseudocode to find the name stored in Name
```

- Bubble sort

- Used to arrange elements in a data structure in ascending or descending order
- Always has a variable to temporarily store data in between swaps
- Compares each element with the next consecutive one, and swaps if needed

Example:

```

DECLARE StudentAge : ARRAY [1:10] OF INTEGER
Last ← 10
FOR Counter ← 1 TO 10
    OUTPUT "Enter student's age"
    INPUT Age
    StudentAge[Counter] ← Age
NEXT Counter
REPEAT
    Swap ← FALSE
    FOR Counter ← 1 TO Last - 1
        IF StudentAge[Counter] > StudentAge[Counter + 1]
            THEN
                Temp ← StudentAge[Counter]
                StudentAge[Counter] ← StudentAge[Counter + 1]
                StudentAge[Counter + 1] ← Temp
            ENDIF
    NEXT Counter
    Last ← Last - 1
UNTIL (NOT Swap) OR Last = 1
//pseudocode to arrange the students' ages in StudentAge in
//ascending order

```

- Totalling

- The process of keeping a running total of values in a program

Examples:

```

TotalCost ← TotalCost + FurnitureCost
TotalBaskets ← TotalBaskets + BananaBaskets

```


- Counting

→ Number of times an action is performed in a program

Examples:

```
Counter ← Counter + 1
```

```
NumberInStock ← NumberInStock - 1
```

- Maximum, minimum, and average

→ Maximum value is found out by comparing all the values input in the program and identifying the largest one

→ Minimum value is found by comparing all the values input in a program and identifying the smallest one

→ Average (mean) value is found by totalling all the values, and dividing that total with the number of values added

Validation

→ The checking that the data entered is reasonable/sensible

→ There are several kinds of validation checks

Validation check	Function (checks if:)	Example
Range	The data is within a set range	REPEAT OUTPUT "Enter your marks" INPUT Marks UNTIL Marks > -1 AND Marks < 101
Length	The data is either an exact number of characters, or within a range of number of characters	REPEAT OUTPUT "Enter your password" INPUT Password UNTIL LENGTH>Password) >= 8
Type	The data conforms to the required data type	REPEAT OUTPUT "Enter your age" INPUT Age UNTIL DIV(Age, 1) = Age

		<code>//This may not be needed as the data type //of Age can be declared INTEGER</code>
Presence	Data has been entered	<code>REPEAT OUTPUT "Enter your name" INPUT Name UNTIL Name <> " "</code>
Format	The data conforms to a specific format	<code>REPEAT OUTPUT "Enter your device code" INPUT DeviceCode UNTIL SUBSTRING(DeviceCode, 1, 3) = "XYX" AND LENGTH(DeviceCode) = 6</code>
Check digit	The final digit entered in a code is correct	<code>REPEAT OUTPUT "Enter your device number" INPUT DeviceNumber OUTPUT "Enter your device code" INPUT DeviceCode UNTIL SUBSTRING(DeviceCode, 6, 1) = (2 * DeviceNumber) //DeviceCode contains a six-digit number</code>

Verification

- The checking if data has been accurately copied from one medium to another
- Does not check for any ranges, boundaries, etc.
- Only checks if the data is identical to the original source

Verification check	Function	Example
Double entry	Requires data to be entered twice so that both entries can be compared to confirm that they are the same	<code>REPEAT OUTPUT "Enter your password" INPUT Password1 OUTPUT "Enter your password again" INPUT Password2 UNTIL Password1 = Password2</code>
Visual	Manual check done by the user to see if the data entered is correct	Data is displayed on the screen and user is asked to confirm their accuracy

Test Data

→ Data that has been specifically identified for testing a program

→ Used to work through a program to find any errors and ensure that it is working like it is meant to

Normal test data: Data that will be accepted by the program

Abnormal/Erroneous test data: Data that will be rejected by the program

Boundary test data: The largest/smallest accepted values, and their corresponding largest/smallest rejected values

Extreme data: The largest and smallest accepted values

UNIT 8: Pseudocode

(NOTE: meta-variables, i.e: symbols in the pseudocode that should be substituted with other symbols, are enclosed in angled brackets <>)

(NOTE: All the examples are in PSEUDOCODE. None of the programming languages are used in these notes.)

- Variable declaration

Format	DECLARE <identifier> : <data type>
---------------	------------------------------------

Examples:

DECLARE MyName : STRING DECLARE Flag : BOOLEAN

- Constant declaration

→ It is recommended to use constants if it makes the program more readable, and easier to update if the value of the constant changes

→ Only literals must be used as the value of a constant; a variable, another constant, or an expression must never be assigned to a constant

Format	CONSTANT <identifier> ← <value>
---------------	---------------------------------

Example:

CONSTANT StudentNo ← 30 CONSTANT Message ← "You have completed this level"

- Identifiers

- Names given to variables, constants, procedures, data structures, and functions
- Must be meaningful, and not arbitrary
- Are not case sensitive; for example, `StudentNumber` and `Studentnumber` must not be treated as different variables
- Keywords should never be used as identifier names, like `Repeat` or `Procedure`
- Should never start with a number e.g: `4Digit`
- Should not include accented letters and other characters, including the underscore (`_`)
- Can only contain uppercase (A-Z) and lowercase (a-z) alphabets, and the digits 0-9
- Must start with a capital letter

Data types

Keywords	Type of data	Literals (examples)	Notes
INTEGER	A whole number	4, 10, 69	Written normally, like in the denary system
REAL	A number capable of containing a fractional part	4.0, 10.8, 69.69	Always written with at least one digit on either side of the decimal point, zero being added if required
BOOLEAN	The logical values TRUE and FALSE	TRUE, FALSE	—
CHAR	A single character	'X', 'M'	Enclosed in single quotes "
STRING	A sequence of zero or more characters	"Hello, how are you?" "" (this is an empty string)	Enclosed in double quotes "" Can contain no characters

- Sequence

- The order in which statements in a program are executed
- Sequence is very important, as an incorrect order can lead to errors

- Selection

- Performed when different actions need to be performed based on the values entered into the algorithm
- There are two types of selection statements:
 - ↳ IF statements
 - ↳ CASE statements
- Also called conditional statements

- IF statements

- IF...THEN...ELSE...ENDIF
- THEN path is followed if the set condition evaluates to TRUE
- ELSE path is followed if the set condition evaluates to FALSE
- May have more than two conditions
- May or may not have an ELSE clause

	Format	Example
Without ELSE clause	<pre>IF <condition> THEN <statements> ENDIF</pre>	<pre>IF Number < 0 THEN OUTPUT "It is a negative number" ENDIF</pre>
With ELSE clause	<pre>IF <condition> THEN <statements> ELSE <statements> ENDIF</pre>	<pre>IF Number < 0 THEN OUTPUT "It is a negative number" ELSE OUTPUT "It is a positive number" ENDIF</pre>
Nested IF statements	<pre>IF <condition> THEN <statements> ELSE IF <condition> THEN <statements> ELSE <statements> ... ENDIF ENDIF (Note that there can be more than one nested IF statements)</pre>	<pre>IF ChallengerScore > ChampionScore THEN IF ChallengerScore > HighestScore THEN OUTPUT ChallengerName, " is champion and highest scorer" ELSE OUTPUT Player1Name, " is the new champion" ENDIF ELSE OUTPUT ChampionName, " is still the champion" IF ChampionScore > HighestScore THEN OUTPUT ChampionName, " is also the highest scorer" ENDIF ENDIF</pre>

- CASE statements

- CASE OF...OTHERWISE...ENDCASE
- Allow one of the several branches of the code to be executed, depending on the value of a variable
- OTHERWISE is used as the last case as the path to be followed if all preceding cases evaluate to FALSE

Format	Example
<pre> CASE OF <identifier> <case 1> : <statements> <case 2> : <statements> <case 3> : <statements> ... OTHERWISE <statements> ENDCASE </pre> <p>(Note that a CASE statement can have any number of cases)</p>	<pre> CASE OF Week 1 : OUTPUT "Monday" 2 : OUTPUT "Tuesday" 3 : OUTPUT "Wednesday" 4 : OUTPUT "Thursday" 5 : OUTPUT "Friday" 6 : OUTPUT "Saturday" 7 : OUTPUT "Sunday" OTHERWISE OUTPUT "Invalid input" ENDCASE </pre>

- Iteration

- When part of the code needs to be repeated either a set number of times or until a set condition evaluates to TRUE or FALSE
- There are three types of iteration loops
 - count-controlled loops
 - pre-condition loops
 - post-condition loops

- Count-controlled loops

- FOR...TO...NEXT...
- Used when there is a known, set number of repetitions
- More efficient than the other loops as the counter variable does not need to be managed/the counter variable increments by itself

	Format	Example
When the counter variable only needs to be incremented by one	<pre>FOR <identifier> ← <value1> TO <value2> <statements> NEXT <identifier></pre>	<pre>FOR Counter ← 1 TO 10 OUTPUT "***" NEXT Counter //this loop prints three *'s //each on ten lines</pre>
When the counter variable needs to be incremented by a specific number	<pre>FOR <identifier> ← <value1> TO <value2> STEP <increment> <statements> NEXT <identifier></pre>	<pre>FOR Counter ← 2 TO 100 STEP 2 OUTPUT Counter NEXT Counter //This loop outputs all even //numbers between 1 and 100 //inclusive</pre>

- Pre-condition loops

- WHILE...DO...ENDWHILE
- Used when there is no known number of repetitions
- Tests the condition prior to the execution of the statements for each repetition
- If it evaluates as TRUE, the statements are executed; otherwise, the loop is terminated
- May not be executed even once if the condition evaluates as FALSE when it is tested for the first time
- The condition must be an expression that evaluates to a Boolean

Format	Example
<pre>WHILE <identifier> DO <statements> ENDWHILE</pre>	<pre>WHILE Number <> 0 DO Total ← Total + Number ENDWHILE //This loop totals all the numbers input, as long //as they are not zero</pre>

- Post-condition loops

- REPEAT...UNTIL
- Used when there is no known number of repetitions
- Tests the condition after the execution of the statements for each repetition
- If the condition is evaluated as FALSE, the statements are executed for the next repetition; otherwise, the loop is terminated
- Is always repeated at least once

Format	Example
REPEAT <statements> UNTIL	REPEAT INPUT Number UNTIL Number < 0 OR Number >100 //This loop repeats until the number input is smaller //than 0 or larger than 100

- String handling

- **Length** : Finding the number of characters in a string
- **Substring** : Extracting a part of a string
- **Uppercase** : Converting all the letters in the string to uppercase
- **Lowercase** : Converting all the letters in the string to lowercase
- The first character of a string can be in position one or zero; however, it is generally one

	Format	Example	Notes
Length	LENGTH ("<string>") OR LENGTH (<identifier>)	LENGTH ("Mojza") OR Name ← "Mojza" LENGTH (Name)	A string in double quotes or a variable with data type string can be used
Substring	SUBSTRING ("<string>", <start>, <length>) OR SUBSTRING (<identifier>, <start>, <length>)	SUBSTRING("Mojza", 1, 3) OR SUBSTRING (Name,1, 3) (This would extract 'Moj')	The first parameter can be a string in double quotes or a variable with data type string



			The second parameter is the position of the start character The third parameter is the length of the substring
Uppercase	UCASE (" <code><string></code> ") OR UCASE (<identifier>)	UCASE ("Mojza") OR UCASE (Name) (This would return "MOJZA")	A string in double quotes, char in single quotes or a variable with data type string or char can be used
Lowercase	LCASE (" <code><string></code> ") OR LCASE (<identifier>)	LCASE ("Mojza") OR LCASE (Name) (This would return 'mojza')	A string in double quotes, char in single quotes, or a variable with data type string or char can be used

- Arithmetic operators

Operator	Function
+	Add
-	Subtract
*	Multiply
/	Divide
^	Raise to the power of
()	Group
MOD	Give the remainder of a division
DIV	Give the quotient of a division

- Logical operators

Operator	Comparison
>	Greater than
<	Lesser than
>=	Greater than or equal to
<=	Lesser than or equal to
<>	Not equal to
=	Equal to

- Boolean operators

Operator	Meaning
AND	Both
NOT	Not
OR	Either

- Procedures

- A standard subroutine that does not return a value
- Can have up to two parameters in our syllabus
- A parameter is a value sent to the subroutine
- The variables declared and used within it are local variables; their scope covers only the procedure
- Value is not returned

- Procedures without parameters:

Format
<pre> PROCEDURE <identifier> <statements> ENDPROCEDURE CALL <identifier> //calling the procedure </pre>

Example

```

PROCEDURE Welcome
    OUTPUT "Welcome back to the game, player!"
    OUTPUT "Click SKIP to skip the tutorial"
ENDPROCEDURE

CALL Welcome

```

- Procedures with parameters:

Format

```

PROCEDURE <identifier>(<param1> : <data type>, <param2> : <data
type>...)
    <statements>
ENDPROCEDURE

CALL <identifier> (Value1, Value2,...)
//calling the procedure

```

Example

```

PROCEDURE Welcome (Name : STRING, Score : INTEGER)
    OUTPUT "Welcome back, ",Name,"!"
    OUTPUT "Your current score is: ", Score
ENDPROCEDURE

CALL Welcome ("EvilDestroyer26", 42000)

```

- Functions

- A standard subroutine that always returns a value
- Can have up to two parameters in our syllabus
- The variables declared and used within it are local variables; their scope covers only the function
- Since functions return a value when they are called; a function call is not a complete programming statement
- It is called with an expression

- Functions without parameters:

Format

```
FUNCTION <identifier> RETURNS <data type>
    <statements>
ENDFUNCTION

<expression> <identifier>
//calling the function
```

Example

```
FUNCTION One RETURNS INTEGER
    One ← 1
    RETURN One
ENDFUNCTION

OUTPUT One()
//calling the function
```

- Functions with parameters:

Format

```
FUNCTION <identifier> (<param1> : <data type>, <param2> : <data
type>...) RETURNS <data type>
    <statements>
ENDFUNCTION

<expression> <identifier> (param1, param2)
//calling the function
```

Example

```
FUNCTION Average (Totalmark : INTEGER, NoOfSubs : INTEGER) RETURNS
REAL
    RETURN TotalMark/NoOfSubs
ENDFUNCTION

OUTPUT "The average mark is ", Average (120, 8)
```

- Local and global variables

- Local variables are variables that are declared for use in a specific *part* of a program
- Their scope is restricted to that part of the program
- Examples of local variables include those which have been declared in a procedure and/or function
- Global variables are variables that are declared for use all over a program
- Their scope is not restricted to any specific part of the program

- Library routines

- Standard subroutines that are available for immediate use
- Some examples of library routines are MOD, DIV, ROUND, and RANDOM
- Identifiers are of integer data type in MOD and DIV

Library routine	Format	Example
DIV	DIV(<identifier1>,<identifier2>) Returns the quotient from the result of the division of identifier1 by identifier2	DIV(16, 5) //Returns 3 DIV(20, 3) //Returns 6
MOD	MOD(<identifier1>,<identifier2>) Returns the remainder from the result of the division of identifier 1 by identifier2	MOD(16, 5) //Returns 1 MOD(20, 3) //Returns 2
ROUND	ROUND(<identifier, <places>) Rounds the identifier to <places> number of decimal places	ROUND(3.1415, 2) //Returns 3.14 ROUND(6.98743, 0) //Returns 7
RANDOM	RANDOM() Returns a number between 0 and 1 inclusive	RANDOM() //Can return 0.11 RANDOM() * 8 //Can return 4.69; //returns a number //between 0 and 8 //inclusive

- Comments

- Are important for making program code more understandable
- Normally, comments are on a separate line before, and at the same level of indentation as, the code they refer to
- Occasionally, however, a short comment that refers to a single line may be at the end of the line to which it refers
- In pseudocode, comments are preceded by two forward slashes //
- The comment continues until the end of the line
- For multi-line comments, each line is preceded by //

Example:

```
// This procedure swaps
// values of X and Y
PROCEDURE SWAP (X : INTEGER, Y : INTEGER)
    Temp ← X // temporarily store X
    X ← Y
    Y ← Temp
ENDPROCEDURE
```

Arrays

- Fixed-length structures of elements of the same data types
- The elements are accessible by consecutive index numbers
- The lower bound (i.e: the index of the first element) can be one or zero
- However, it is generally one
- The upper bound (i.e: the index of the last element) of the array can be any integer

- 1-D arrays

- Declaration:

Format

```
DECLARE <identifier> : ARRAY [<l>:<u>] OF <data type>
```

l is the lower bound of the array and u is its upper bound

Example

```
DECLARE Name : ARRAY [1:10] OF STRING
```

- Input/Output values:

Input

```
Name [2] ← "Bruce Wayne"
```

Output

```
OUTPUT Name[2]
```

- Initialisation/assignment of values:

```
FOR Counter ← 1 TO 10  
    Name[Counter] ← ""  
NEXT Counter
```

- 2-D arrays

- Declaration:

Format

```
DECLARE <identifier> : ARRAY [<l1>:<u1>, <l2>:<u2>] OF <data type>
```

l1 is the lower bound of the number of rows in the array, and u1 is its upper bound

l2 is the lower bound of the number of columns in the array, and u2 is its upper bound

Example

```
DECLARE StudentMark : ARRAY [1:ClassSize, 1:SubjectNo] OF REAL
```


- Input/Output values:

Input

```
StudentMark [3,5] ← 45.0
```

Output

```
OUTPUT StudentMark [3,5]
```

- Initialisation/assignment of values:

```
FOR RowCounter ← 1 TO ClassSize
  FOR ColumnCounter ← 1 TO SubjectNo
    StudentMark [RowCounter : ColumnCounter] ← 0.00
  NEXT ColumnCounter
NEXT RowCounter
```

Storing data in a file

- Data is stored in files in programs for multiple reasons
- Data stored in files is not lost when the computer is switched off/it is stored permanently
- It can be used by more than one program or reused when a program is run again
- It can be backed up or archived

File Handling

- Before reading from or writing to a file, explicitly opening it and stating the mode of operation is a good practice
- There are two modes of operation:
 - ↳ READ : for data to be read from a file
 - ↳ WRITE : for data to be written to a file
- A file can only be opened in one mode at a time

READ

Format

```
OPENFILE <File identifier> FOR READ  
READFILE <File identifier>, <Variable>  
CLOSEFILE <File identifier>
```

Example

```
OPENFILE MyFile.txt FOR READ  
READFILE MyFile.txt, LineOfText  
CLOSEFILE MyFile.txt
```

WRITE

Format

```
OPENFILE <File identifier> FOR WRITE  
WRITEFILE <File identifier>, <variable>  
CLOSEFILE <File identifier>
```

Example

```
OPENFILE MyFile.txt FOR WRITE  
WRITEFILE MyFile.txt, LineOfText  
CLOSEFILE MyFile.txt
```

Unit 9: Databases

Databases:

- A database is a structured collection of data that allows people to extract information in a way that meets their needs.
- Data can include text, images, numbers.
- Single-table database contains only one table

- Can store information about people, products, events, timings and more

- Data is stored in tables, which further consists of records.
- Records further consist of fields. Number of records can vary as data is entered or deleted.
- Number of fields is fixed
- Records are rows
- Fields are columns

TABLE						
	Record 1	Field 1	Field 2	Field 3	Field 4	Field 5
	Record 2	Field 1	Field 2	Field 3	Field 4	Field 5
Row	Record 3	Field 1	Field 2	Field 3	Field 4	Field 5
	Record 4	Field 1	Field 2	Field 3	Field 4	Field 5
	Record 5	Field 1	Field 2	Field 3	Field 4	Field 5
				Column		

- The table has a fixed name, such as TICKETS (this table is for bus tickets)
- Each record will be different with fields
- Each field will have different data such as FirstName, LastName, TicketNo, BusNo, and more. There will be *no spaces* in between

→ Example of a table:

TICKETS					
TicketNo	FirstName	LastName	BusNo	Price	Destination
TicketNo #1	FirstName #1	LastName #1	BusNo #1	Price#1	Destination #1
TicketNo #2	FirstName #2	LastName #2	BusNo #2	Price#2	Destination #2
TicketNo #3	FirstName #3	LastName #3	BusNo #3	Price#3	Destination #3
TicketNo #4	FirstName #4	LastName #4	BusNo #4	Price#4	Destination #4
TicketNo #5	FirstName #5	LastName #5	BusNo #5	Price#5	Destination #5

→ Databases will also use validation checks when data is being entered

→ Validation checks of presence check, range check, type check, length check can be done

→ Each field will have a specific data type and will only accept the data given in that specific data type

→ There are 6 basic data types used in databases:

Data type	Used for
Alphanumeric / text	A number of characters/a mix of alphabets and numbers
Character	A single character
Boolean	TRUE or FALSE / 1 or 0 / YES or NO (for two options)
Integer	A whole number
Real	A decimal number
Date/time	Date and/or time

→ To identify a specific item, person or etc., a unique field is required

→ Unique here means no repetition of data

→ The field that is unique is called a Primary Key

→ In the database table, TICKETS, the primary key can be the TicketNo field, as it has no repetition of data and will be unique

TICKETS					
TicketNo	FirstName	LastName	BusNo	Price	Destination
BS405	Lara	Lee	BS12	20.50	New York
QS564	James	Benjamin	QS11	40.00	Las Vegas
RT567	Louis	Martin	RT10	32.99	Queens
BS506	James	Lee	BS12	20.50	New York
QT234	Walter	White	QT09	15.50	Florida

SQL

- SQL stands for Standard Query Language
- It is for writing scripts to obtain useful information from a database and display it

- SELECT → selects the fields that information needs to be displayed from
- SELECT * → means that all fields are to be displayed
- FROM → specifies the table in which that specific field is present
- WHERE → selects all records with specific condition or data type
- ORDER BY → displays data either in ascending or descending order
- SUM → takes the sum of a specific field, if the field is of real or integer data type
- COUNT → counts how many times the record and field contains the data according to the conditions given

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or 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 where all conditions must be true
OR	Specify multiple conditions where one or more conditions must be true
NOT	Specify a condition that could be false

- An SQL command starts with the SELECT function
- This is followed by the FROM function
- The script then has the commands needed to obtain the information
- The last command of the script is ended with a semicolon

Example 1:

The following SQL query displays the first name of all people going to New York or Florida in alphabetical order, their last names, and their ticket number from the table TICKETS.

```
SELECT FirstName, LastName, TicketNo
FROM TICKETS
WHERE Destination = 'New York' OR 'Florida'
ORDER BY FirstName;
```

This returns:

James	Lee	BS506
Lara	Lee	BS405
Walter	White	QT234

Note that the order in which the script has asked for the information (in this case, first name, last name, and then ticket number) is very important, and so, it must be displayed according to it

Example 2:

The following SQL query displays the prices of tickets, in descending order, for buses that are either going to New York, Queens or Las Vegas, their bus numbers, and their respective destinations.

```
SELECT Price, BusNo, Destination  
FROM TICKETS  
WHERE Destination = 'New York' OR 'Queens' OR 'Las Vegas'  
ORDER BY Price DESC;
```

This returns:

40.00	QS11	Las Vegas
32.99	RT10	Queens
20.50	BS12	New York
20.50	BS12	New York

Unit 10: Boolean Logic

→ There are 6 different logic gates given in 2023, 2024 and 2025 syllabus:



NOT gate



AND gate



OR gate



NAND gate



NOR gate



XOR gate

Truth Tables:

- Used to trace the output from a logic gate or circuit
- Each logic gate is currently restricted to two inputs (except NOT, which has only one)
- Each input will give a different output, based on the logic gate

Logic gates:

1) NOT gate

- The output is the opposite of the input
- $X = \text{NOT } A$ (logic notation)

Input	Output
0	1
1	0

2) AND gate

- Output is 1 when BOTH inputs are 1
- $X = A \text{ AND } B$ (logic notation)

Input A	Input B	Output X
0	0	0
0	1	0
1	0	0
1	1	1

3) OR gate

- When either or both input/s is 1, the output will be 1
- $X = A \text{ OR } B$

Input A	Input B	Output X
0	0	0
0	1	1
1	0	1
1	1	1

4) NAND gate

- Opposite of AND gate
- Output will be 1 when both inputs are NOT 1
- $X = A \text{ NAND } B$

Input A	Input B	Output X
0	0	1
0	1	1
1	0	1
1	1	0

5) NOR gate

- Opposite of OR gate
- Output will be 1 when 1 is NOT an input
- $X = A \text{ NOR } B$

Input A	Input B	Output X
0	0	1
0	1	0
1	0	0
1	1	0

6) XOR (EOR) gate

- Output is 1 when both inputs are NOT the same
- $X = A \text{ XOR } B$

Input A	Input B	Output X
0	0	0
0	1	1
1	0	1
1	1	0

A truth table with three inputs A, B AND C looks like this

:

A	B	C
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Make a logic circuit from:

- A problem statement

- Form a logic expression from the problem statement
- Draw the logic circuit

Example:

A gas fire has a safety circuit made up of logic gates. It generates an alarm ($X = 1$) in response to certain conditions.

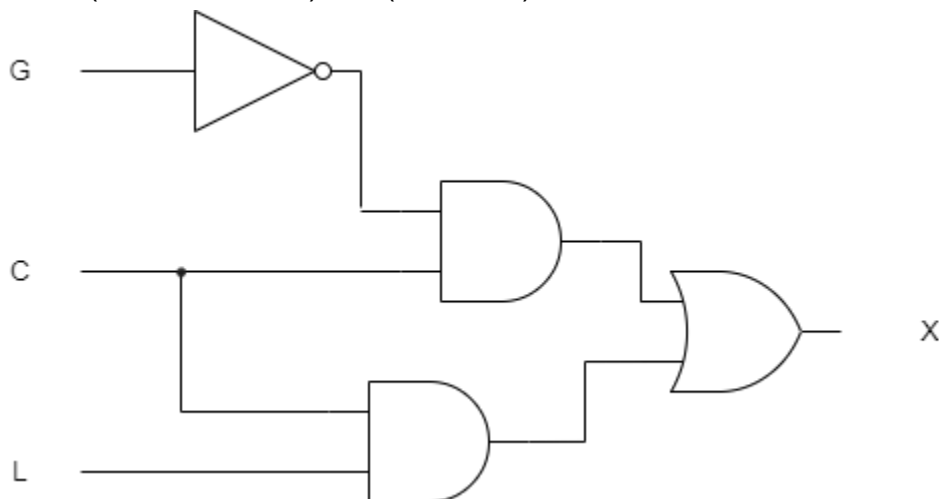
Input	Description	Binary value	Condition
G	Gas pressure	0	Gas pressure is correct
		1	Gas pressure is too high
C	Carbon monoxide level	0	Carbon monoxide level is correct
		1	Carbon monoxide level is too high
L	Gas leak detection	0	No gas leak is detected
		1	Gas leak is detected

The output $X = 1$ is generated under the following conditions:

Gas pressure is correct **AND** carbon monoxide level is too high
OR
carbon monoxide level is correct **AND** gas leak is detected

Solution:

X is 1 if: $(\text{NOT } G \text{ AND } C) \text{ OR } (C \text{ AND } L)$



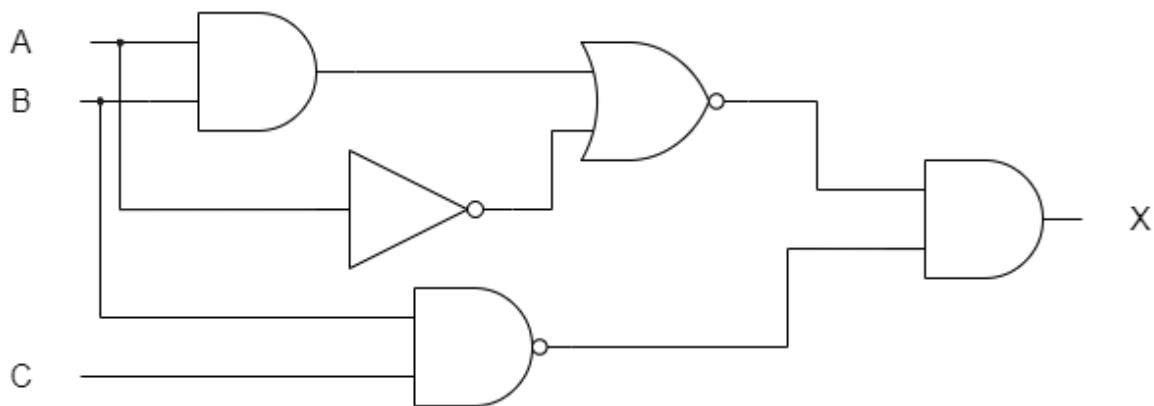
- A logic expression:

→ Identify the innermost pair of brackets, and work your way out of it to draw the logic circuit properly

Example:

$((A \text{ AND } B) \text{ NOR } (\text{NOT } A)) \text{ AND } (B \text{ NAND } C)$

Solution:



- A truth table:

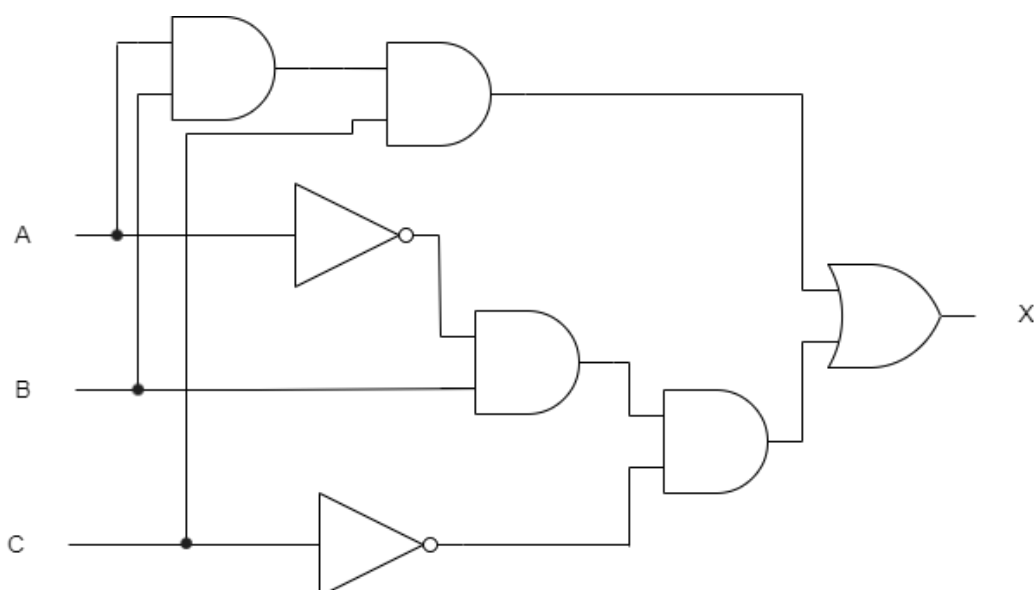
- Identify the rows where the output is 1
- Form a logic expression
- Draw the logic circuit

Example:

A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Solution:

- Identify the rows where the output is 1
- Write each row's expression:
 - Row 3: (NOT A AND B) AND (NOT C)
 - Row 8: (A AND B) AND C
- Now, form a full logic expression by connecting each part with OR
- Add brackets as required
- $((\text{NOT } A \text{ AND } B) \text{ AND } (\text{NOT } C)) \text{ OR } ((A \text{ AND } B) \text{ AND } C)$



Complete a truth table from:

- A problem statement

- Make the logic expression for the problem statement
- Make a truth table with three inputs and complete it

Example:

A gas fire has a safety circuit made up of logic gates. It generates an alarm ($X = 1$) in response to certain conditions.

Input	Description	Binary value	Condition
G	Gas pressure	0	Gas pressure is correct
		1	Gas pressure is too high
C	Carbon monoxide level	0	Carbon monoxide level is correct
		1	Carbon monoxide level is too high
L	Gas leak detection	0	No gas leak is detected
		1	Gas leak is detected

The output $X = 1$ is generated under the following conditions:

Gas pressure is correct **AND** carbon monoxide level is too high
OR
carbon monoxide level is correct **AND** gas leak is detected

Solution:

X is 1 if : (NOT G AND C) OR (C AND L)

G	C	L	Working Space			X (Q OR R)
			NOT G (P)	P AND C (Q)	C AND L (R)	
0	0	0	1	0	0	0
0	0	1	1	0	0	0
0	1	0	1	1	0	1
0	1	1	1	1	1	1
1	0	0	0	0	0	0
1	0	1	0	0	0	0
1	1	0	0	0	0	0
1	1	1	0	0	1	1

You must show your working in the provided working space, and you can name each column an alphabet (like 'P AND C' was named 'Q' above) for your ease.

- A logic expression:

- Make the logic expression for the problem statement
- Make a truth table with three inputs and complete it

Example:

((A AND B) NOR (NOT A)) AND (B NAND C)

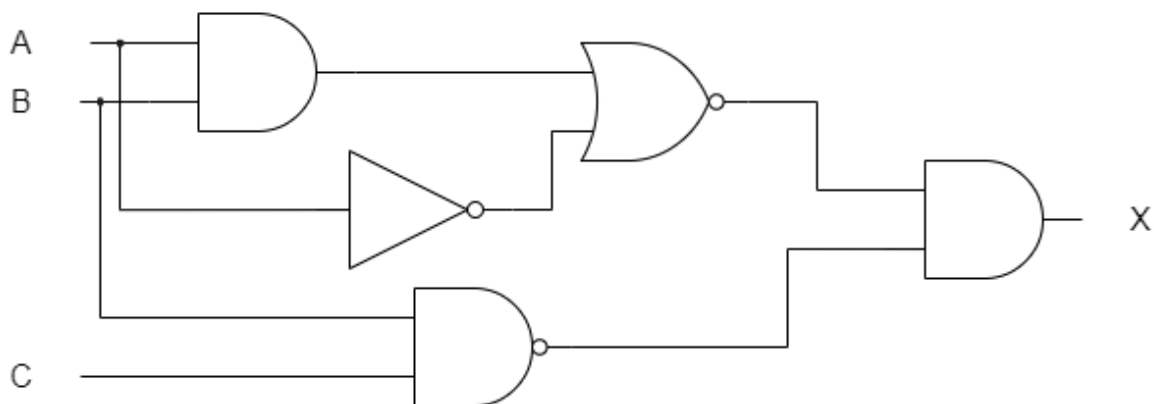
Solution:

A	B	C	Working Space				X (F AND G)
			A AND B (D)	NOT A (E)	D NOR E (F)	B NAND C (G)	
0	0	0	0	1	0	1	0
0	0	1	0	1	0	1	0
0	1	0	0	1	0	1	0
0	1	1	0	1	0	0	0
1	0	0	0	0	1	1	1
1	0	1	0	0	1	1	1
1	1	0	1	0	0	1	0
1	1	1	1	0	0	0	0

- A logic circuit:

- In the working space, name each part of the circuit with an alphabet for your ease
- Make the truth table and complete it

Example:



Solution:

A	B	C	Working Space				X (F AND G)
			A AND B (D)	NOT A (E)	D NOR E (F)	B NAND C (G)	
0	0	0	0	1	0	1	0
0	0	1	0	1	0	1	0
0	1	0	0	1	0	1	0
0	1	1	0	1	0	0	0
1	0	0	0	0	1	1	1
1	0	1	0	0	1	1	1
1	1	0	1	0	0	1	0
1	1	1	1	0	0	0	0

Write a logic expression from:

- A problem statement

→ Identify the conditions

→ Write their logical equivalents

Example:

A gas fire has a safety circuit made up of logic gates. It generates an alarm ($X = 1$) in response to certain conditions.

Input	Description	Binary value	Condition
G	Gas pressure	0	Gas pressure is correct
		1	Gas pressure is too high
C	Carbon monoxide level	0	Carbon monoxide level is correct
		1	Carbon monoxide level is too high
L	Gas leak detection	0	No gas leak is detected
		1	Gas leak is detected

The output $X = 1$ is generated under the following conditions:

Gas pressure is correct **AND** carbon monoxide level is too high
OR
carbon monoxide level is correct **AND** gas leak is detected

Solution:

→ Gas pressure is correct: NOT G (because $G = 0$)

Carbon monoxide level is too high : C (because $C = 1$)

→ Combine these two with AND, adding brackets as required
(NOT G) AND C

→ Carbon monoxide level is correct: NOT C (because $C = 0$)

→ Gas leak is detected: L (because $L = 1$)

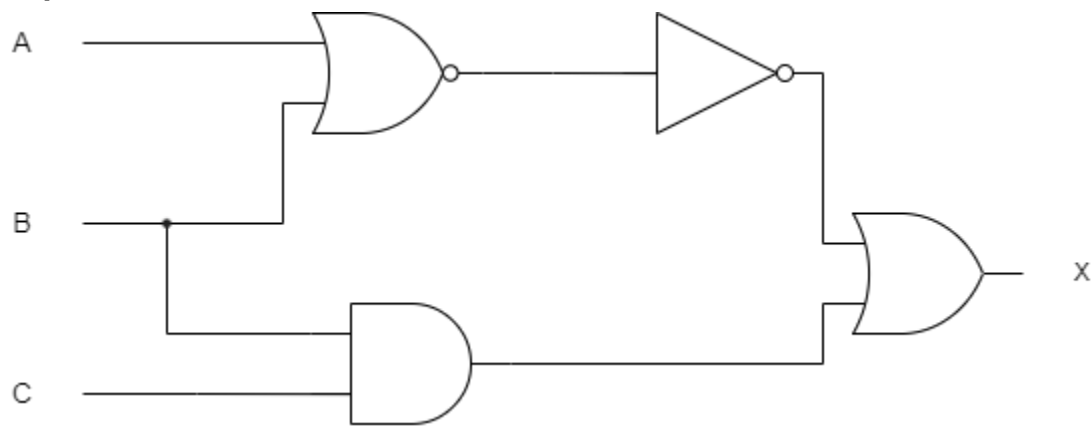
→ Combine these two with AND, adding brackets as required
(NOT C) AND L

→ Finally, combine (NOT G) AND C and (NOT C) AND L with OR
((NOT G) AND C) OR ((NOT C) AND L)

- A logic circuit:

- Separate the logic circuit into each of its sections, going from left to right
- Combine them, using brackets as needed, and form the logic expression

Example:



Solution:

- The first two sections are:

A NOR B

B AND C

- The output of the first section is input to a NOT gate

NOT (A AND B)

- Connect NOT (A AND B) and B AND C using an OR gate

NOT (A AND B) OR B AND C

- Add brackets for accuracy

(NOT (A AND B)) OR (B AND C)

- A truth table:

- Identify the rows which have 1 as an output
- Write their logic expressions
- Combine with OR

Example:

A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Solution:

→ Rows with 1 as an output are highlighted

→ Row 4:

$((\text{NOT } A) \text{ AND } B) \text{ AND } C$

→ Row 6:

$(A \text{ AND } (\text{NOT } B)) \text{ AND } C$

→ Row 8:

$(A \text{ AND } B) \text{ AND } C$

→ Combine all three with OR, adding brackets as required

$((\text{NOT } A) \text{ AND } B) \text{ AND } C) \text{ OR } ((A \text{ AND } (\text{NOT } B)) \text{ AND } C) \text{ OR } ((A \text{ AND } B) \text{ AND } C)$

A Note from Mojza

These notes for Computer Science (2210/0478) have been prepared by Team Mojza, covering the content for GCE O levels and IGCSE 2023-25 syllabus. The content of these notes has been prepared with utmost care. We apologise for any issues overlooked; factual, grammatical or otherwise. We hope that you benefit from these and find them useful towards achieving your goals for your Cambridge examinations.

If you find any issues within these notes or have any feedback, please contact us at support@mojza.org.

Acknowledgements

Authors:

Zoella Ahmad
Fasiha Raza

Proof-readers:

Hadiya Farrukh
Hasan Nawaz

Designers:

Fasiha Raza