



Chapter S1A: Algorithmic Representation

CONTENT

No.	Topic	Syllabus Guide
1	Algorithm	1.1
2	Flowchart	1.1.1 – 1.1.2
3	Pseudo-code	1.1.1
	3.1 Identifiers	
	3.2 Keywords	
	3.3 Comments	
	3.4 Data Types	
	3.5 Variables	
	3.6 Constants	
	3.7 Assignments	
	3.8 Operators	
4	Control Structures	1.1.3
	4.1 Sequence	
	4.2 Selection	
	4.3 Iteration	
5	Modular Design	1.1.5
	5.1 Subroutines	
	5.2 Procedures	
	5.3 Functions	
	5.4 Passed by value and by reference	
6	Decision Table	1.1.4
	6.1 Types of decision tables	
	6.2 Basic format	
	6.3 Rules	
	6.4 Advantages & disadvantages	
	6.5 Decision tree	
	6.6 Removing redundancies	
7	References	

1. Algorithm

An algorithm is a sequence of steps designed to perform a particular task. It is a **well-ordered** collection of **unambiguous** and **effectively computable operations**, when executed, produces a result and halts in a finite amount of time.

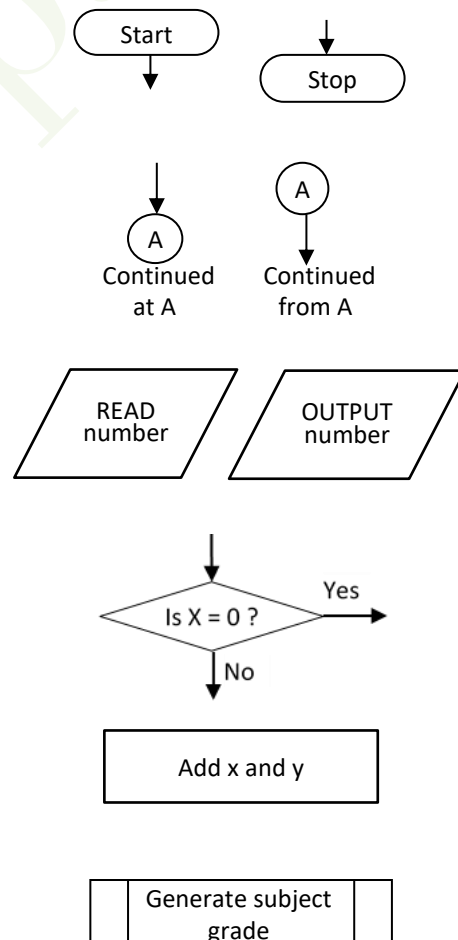
Algorithms solve problems. In order to know whether a solution is correct, **an algorithm must produce an observable result** so that a user can determine whether it is the desired one. The result must be produced after the execution of a **finite number of operations**.

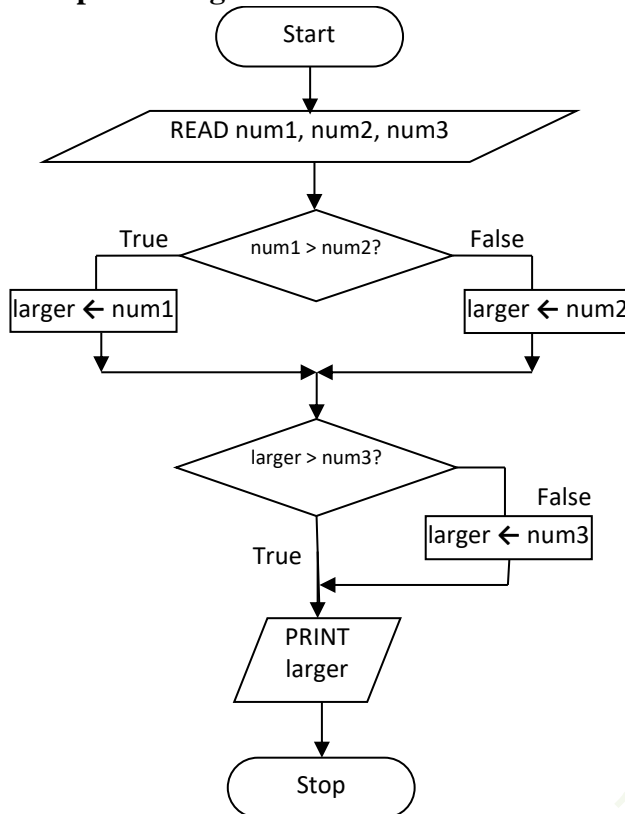
Algorithms can be written in any suitable form, such as a programming language, pseudo-code or as diagrams such as flowchart. Often they are written in pseudo-code, which is easily communicated and easily translated into any suitable programming language.

2. Flowchart

Flowchart is a graphical representation of the operations involved in a process or system. Symbols are used to represent particular operations and flow lines indicate the sequence of operations. Flow lines may use arrows to indicate sequence, or a top-down, left-right convention may apply if there are no arrows. Different shapes indicate the various kinds of activity described by the flowchart.

- **Terminator (Start/Stop)** symbol is used for both starting and stopping points in a flowchart.
- **Connector** symbol is used to indicate that the flowchart is continued elsewhere at an identically labelled connector point.
- **Input/output** symbol is used for any input or output operation.
- **Decision** symbol indicates points in a program where decisions are made.
- **Process** symbol is used for any operation or sequence of instructions that does not involve a decision.
- **Subroutine** symbol is used to indicate a subroutine call. *A subroutine is a set of program instructions performing a specific task, but which is not a complete program.*



Example: An algorithm as a flowchart

What does this algorithm do?

3. Pseudo-code

Pseudo-code is a method of describing an algorithm or program design. It uses *control structures, keywords and identifiers* similar to those found in programming languages, but without following the syntax of a particular programming language. It may be presented in a form that looks like a combination of English and a programming language. While there is no standard way of writing pseudo-code, the logic of the solution should be seen clearly. Hence we will follow a pseudo-code style that is commonly recognised. A programmer using a high-level language should be able to write the program code from the pseudo-code algorithm design.

The pseudo-code for the above flowchart algorithm may look like this:

```

INPUT num1
INPUT num2
INPUT num3
IF num1 > num2
    THEN
        larger ← num1
    ELSE
        larger ← num2
ENDIF
IF larger > num3
    THEN
        OUTPUT larger
    ELSE
        larger ← num3
        OUTPUT larger
ENDIF
  
```

3.1 Identifiers

An **identifier** is a name or label chosen by the programmer to represent an object within a program. The object could be a variable, a function, a procedure, a data type or any other element defined within a program. They can contain letters, digits and the underscore “_” symbol. It should not begin with a digit. It should be named as meaningfully as possible by using more descriptive words. Single letters may be used where these are conventional, such as *i* and *j* when dealing with array indices, or *X* and *Y* when dealing with coordinates. Keywords are not allowed as identifiers.

Valid identifiers	Invalid identifiers (reason)
name	24S24 (starts with a digit)
bookList	email@ddress (contains invalid symbol @)
speed_of_boat	break (is a keyword)

3.2 Keywords

Keywords are reserved names used to recognise the structure of a programming language. It has its own meaning that is defined in the language and cannot be used as identifiers. For example, `break` is a reserved word used to terminate a loop in many programming languages, and cannot be used as a variable or other identifier.

3.3 Comments

Comments are used to make the algorithm or program code easier to understand. They have no effect on the running of the program. They are also important to document the program code, specifying the name of program, author, date written, purpose of program, and explain how complicated parts of the program work. 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 `//`.

3.4 Data Types

Data types are used to determine the types of values that a variable can have and the operations that can be performed on the variable. They determine the range of values that the memory location can hold.

Data Types	Definition	Literals	Size
INTEGER	a whole number	-33, 0, 8, 79351	4 bytes
REAL	a number with a fractional part, at least one digit on either side of the decimal point	-24.8, 0.0, 3.142, 999.999	4 bytes
CHAR	a single character, such as a letter or number or special character, delimited by single quotes	'a', 'A', '#', '7' (Note that character “7” is represented differently in the computer from the integer 7 or the real number 7.0)	1 byte
STRING	a sequence of zero or more characters, delimited by double quotes	“Programming is my game”	1 byte per character
BOOLEAN	the logical values TRUE and FALSE	TRUE, FALSE	1 byte

3.5 Variables

A variable is a storage location for a data value that has an identifier. In any program input, data has to be stored in computer's memory and accessed for the process of the program. These named memory locations are referred to as variables. It is a good practice to declare variables explicitly in pseudocode.

```
DECLARE age: INTEGER
DECLARE salary: REAL
DECLARE GameOver: BOOLEAN
```

3.6 Constants

Constants hold values that **never change** while the program is run. For example, if your program involves calculating area of a circle, pi can be defined as a constant having the value of 3.14159, or the radius of a circle can be defined as a constant.

```
CONSTANT pi = 3.14159
CONSTANT radius = 3.0
```

Use of constants make the pseudocode more readable, as an identifier is more meaningful than a literal most of the time. It also makes the pseudocode easier to update if the value of the constant changes. Constants are normally declared at the beginning of a pseudocode.

3.7 Assignments

In assignments, a value is given a name (identifier) or the value associated with a given identifier is changed. The assignment operator is \leftarrow .

```
Weight  $\leftarrow$  62.3
Height  $\leftarrow$  1.68
```

3.8 Operators

- Arithmetic operators: Common arithmetic operator symbols are used to manipulate numbers by carrying out normal arithmetic tasks.

+	Addition
-	Subtraction
*	Multiplication
/	Division

- Relational operators: Compare data and produce a result of data type BOOLEAN.

>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
=	Equal to
<>	Not equal to

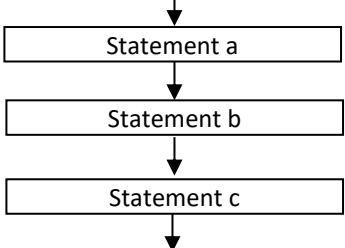
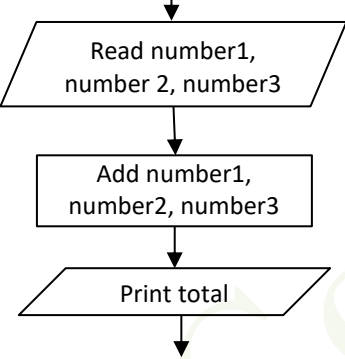
- Logic operators: Include AND, OR, and NOT. The operands and results of these operations are always of data type BOOLEAN.

4. Control Structures (Program constructs)

Control structures manage the flow of the program (i.e.: different ways in which a group of instructions can be executed). They allow the group of instructions to be repeated a fixed or variable number of times, or to be selectively executed depending on the data currently being processed. The three basic control structures are *sequence*, *selection* and *iteration*.

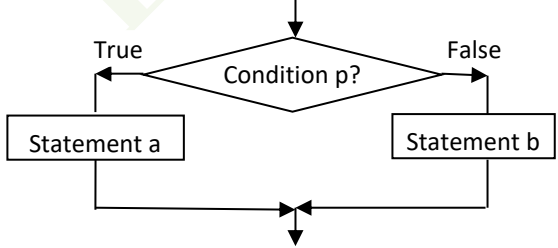
4.1 Sequence

Sequence is the straightforward execution of one processing step after another.

Flowchart	Pseudo-code
 <pre> graph TD Start(()) --> A[Statement a] A --> B[Statement b] B --> C[Statement c] C --> End(()) </pre> <p>Statement a will be executed first, followed by statement b and statement c.</p>	<pre> Statement a Statement b Statement c </pre>
 <pre> graph TD Start(()) --> Read[/Read number1, number 2, number3/] Read --> Add[Add number1, number2, number3] Add --> Print[/Print total/] Print --> End(()) </pre> <p>Read three numbers. Add the three numbers and print their total.</p>	<pre> INPUT number1 INPUT number2 INPUT number3 total ← number1 + number2 + number3 OUTPUT total </pre>

4.2 Selection (conditional transfer)

The **selection control structure** can be defined as the presentation of a condition, and the choice between two actions depending on whether the condition is true or false. It uses an **IF-THEN-ELSE** construct in the pseudo-code.

Flowchart	Pseudo-code
 <pre> graph TD Start(()) --> Cond{Condition p?} Cond -- True --> A[Statement a] Cond -- False --> B[Statement b] A --> Join(()) B --> Join Join --> End(()) </pre> <p>If condition p is true, statement a will be executed. Otherwise statement b will be executed.</p>	<pre> IF Condition p is True THEN Statement a ELSE Statement b ENDIF </pre>

<pre> graph TD Start(()) --> A[ACCEPT ← False] A --> D{LENGTH (TelephoneNumber) = 8?} D -- True --> B[ACCEPT ← True] D -- False --> Exit(()) B --> Exit </pre> <p>TelephoneNumber is fixed at eight digits. If the length of TelephoneNumber is eight, ACCEPT will be set to True, otherwise ACCEPT will be False.</p>	<pre> ACCEPT ← False IF LENGTH(TelephoneNumber) = 8 THEN ACCEPT ← True ENDIF </pre>
---	---

4.3 Iteration (repetition/loop)

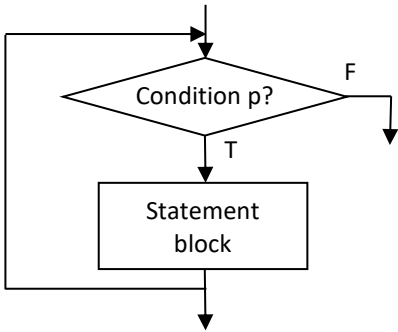
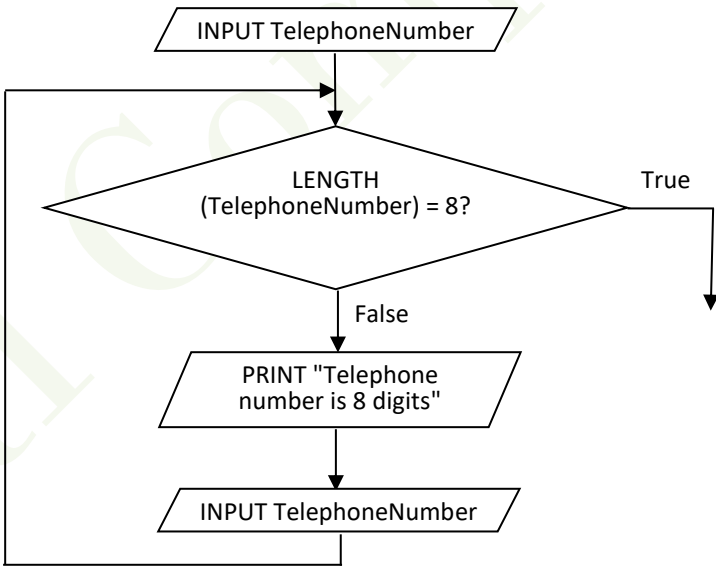
The **iteration control structure** can be defined as the presentation of a set of instructions to be performed repeatedly, as long as a condition is true.

There are three methods for iteration – FOR loop, WHILE loop and REPEAT-UNTIL loop. The choice of method is dependent on the context given.

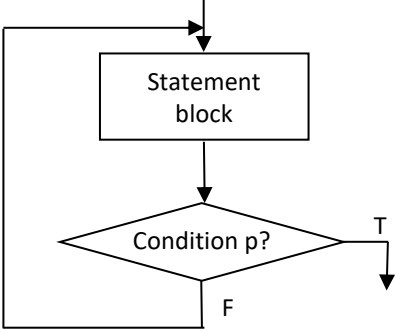
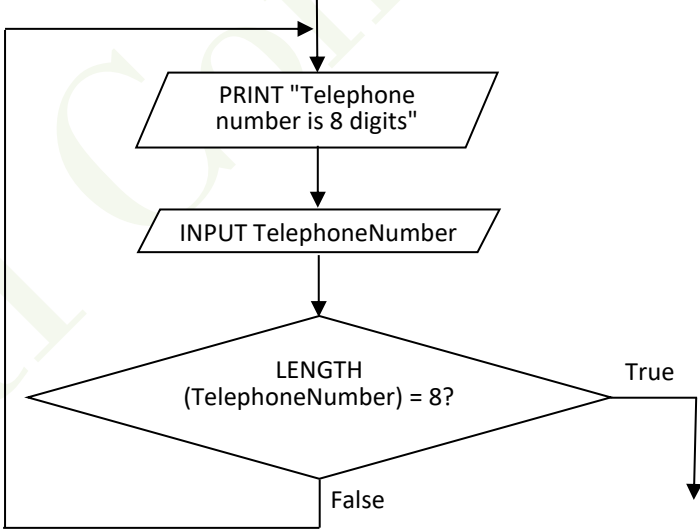
4.3.1 FOR loops are commonly used when the number of iterations is known beforehand. It is also known as a *count-controlled loop* because the loops are executed a fixed number of times.

Flowchart	Pseudo-code
<pre> graph TD Start(()) --> C[Count ← 1] C --> D{Count ≤ 10?} D -- T --> E[Total ← Total + Count] E --> F[Count ← Count + 1] F --> D D -- F --> Exit(()) </pre> <p>Loop is executed _____ times.</p>	<pre> FOR Count ← 1 TO 10 STEP 1 IF Count ≤ 10 THEN Total ← Total + Count ENDIF ENDFOR Count </pre>

4.3.2 WHILE loop is a pre-test loop, it tests the condition *before* entering the loop. Whenever the condition is true, the loop is executed. When the condition is false, the loop exits. If in the first instance the condition is tested and found to be false, the body of the loop will not run at all.

Flowchart	Pseudo-code
 <p>Condition is checked at the of the loop. When condition is True, the loop When condition is False, the loop</p>	<pre> WHILE Condition p is True Statement block ENDWHILE </pre>
	
<pre> INPUT TelephoneNumber WHILE OUTPUT "Telephone number is 8 digits" INPUT ENDWHILE </pre>	

4.3.3 REPEAT-UNTIL loop *executes the body of the loop at least once*, after which the condition is checked. This is a post-test loop, because the condition is tested *after* the body of the loop is executed. *The body of the loop keeps repeating until the condition is true.*

Flowchart	Pseudo-code
 <p>Condition is checked at the of the loop. The body of the loop (statement block) will be executed at least When condition is True, the loop When condition is False, the loop</p>	<pre> REPEAT Statement block UNTIL Condition p is True Alternatively, the while loop may be used: Statement block WHILE Condition p is False Statement block ENDWHILE </pre>
	
<pre> REPEAT OUTPUT "Telephone number is 8 digits" INPUT UNTIL </pre>	

Example: A program is to be written to calculate the discount given on purchases. A purchase may qualify for a discount depending on the amount spent. The purchase price (Purchase), the discount rate (DiscountRate) and amount paid (Paid) are calculated as shown in the following pseudocode algorithm.

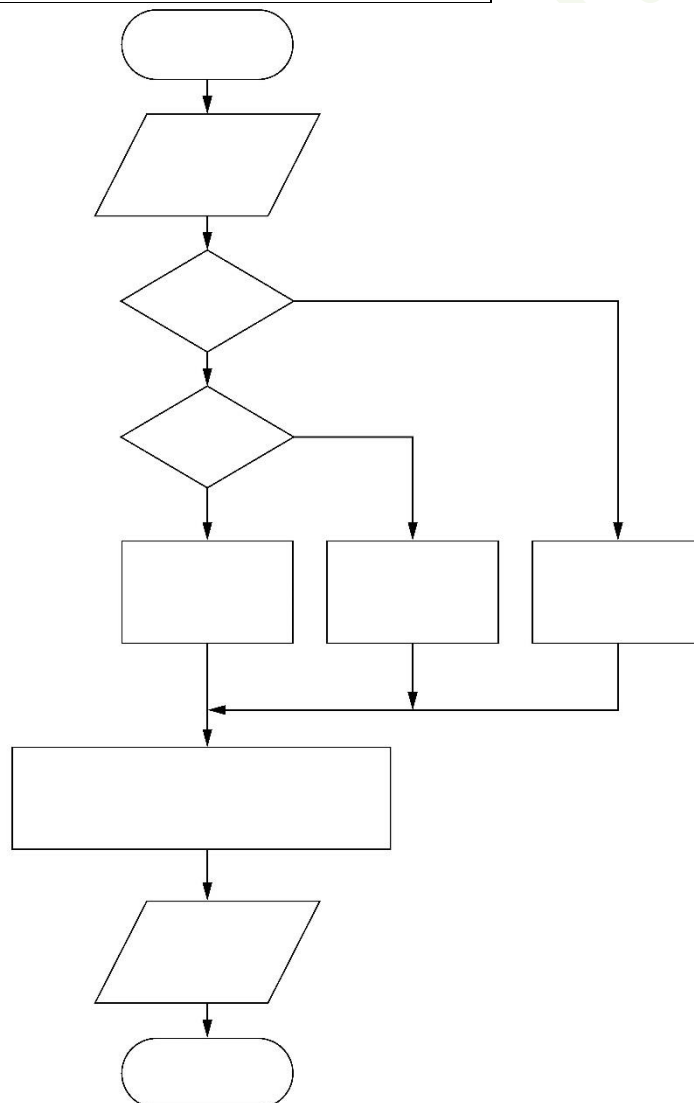
```

INPUT Purchase
IF Purchase > 1000
    THEN
        DiscountRate ← 0.10
    ELSE
        IF Purchase > 500
            THEN
                DiscountRate ← 0.05
            ELSE
                DiscountRate ← 0
        ENDIF
    ENDIF
Paid ← Purchase * (1 - DiscountRate)
OUTPUT Paid

```

The algorithm is also to be documented with a program flowchart.

Complete the flowchart by filling in the flowchart symbols and labelling, where appropriate, the lines of the flowchart.



5. Modular Design

Modular design (or top-down design) is a method of organising a large computer program into self-contained parts, modules, which can be developed simultaneously by different programmers or teams of programmers.

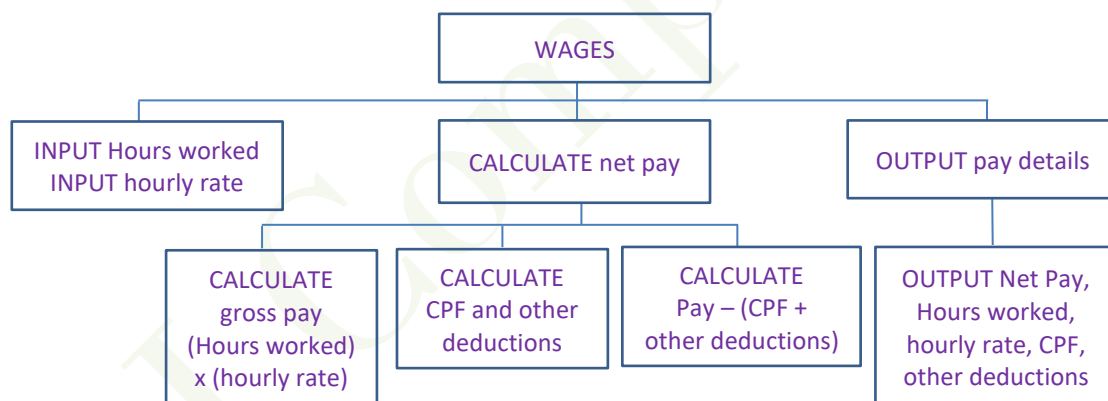
In modular design, start with the original problem and split the problem into smaller and smaller parts until they become manageable. These smaller parts of the solution are known as modules. When each of the individual problems has been solved, the solutions are combined to produce a complete solution to the whole problem.

Modules are typically incorporated into the program through interfaces. A module interface expresses the elements that are provided and required by the module. The elements defined in the interface are detectable by other modules. The implementation contains the working code that corresponds to the elements declared in the interface.

Example:

Given the hourly pay rate and the number of hours worked, calculate the net pay after CPF contribution and other deductions for an employee.

A structure diagram can be used to illustrate modular design. The main module WAGES is broken down into submodules to find the wages of employees.



There are three steps to calculate the net pay. First, calculate the total pay. Next, calculate the CPF and other deductions. Then subtract the CPF and other deductions from the total pay. Output of the pay details are then displayed. These steps are shown as submodules linked to the main module.

As shown from the structure diagram, the calculation of wages can be broken down into modules. These modules are easier to understand than trying to write one program to calculate the wages.

Advantages of modular (top-down) approach:

- *More than one person can be engaged on solving parts of the same problem simultaneously.* A large task is dissected into modules and the task is divided among various people. This enables new programs to be developed faster.
- *The person producing a module is likely to make fewer mistakes,* since the algorithms for each module are much shorter than for the whole problem. It also makes for easier testing of the program code and eliminate errors, thereby improving the reliability of the program.
- *Modules can be reuse* in more than one project, if a similar solution is needed. These modules can be stored and used again if the opportunity arises. Collection of modules like this are known as software or program libraries.

5.1 Subroutines

Modules can be implemented as **subroutines** (or subprogram) which is a named block of code that performs a specific task within a program. A subroutine is not a complete program, it must be incorporated into a program in order to be used.

Most high-level languages support two types of subroutine, **functions** and **procedures**, which are called in a slightly different way. A subroutine is *called by writing its name* in a program statement.

For example, in Python, subroutines **sqrt** and **print** can be called in the following manner:

```
result = sqrt(25)
print("Square root of 25 is", result)
```

A function is called like the **sqrt** above and *always assigns a return value to a variable*. In the above example, a return value which is **5** will be assigned to the variable **result**.

A procedure is called like the **print** above and *does not assign the result to a variable*.

5.2 Procedures

A procedure with no parameters is defined as follows:

```
PROCEDURE <identifier>
    <statement(s)>
ENDPROCEDURE
```

A procedure with parameters is defined as follows:

```
PROCEDURE <identifier>(<param1>:<datatype>,
                        <param2>:<datatype>, ..... )
    <statement(s)>
ENDPROCEDURE
```

The <identifier> is the identifier used to call the procedure. Where used, param1, param2, etc, are identifiers for the parameters of the procedure. These will be used as variables in the statements of the procedure.

Procedures defined as above should be called as follows, respectively:

```
CALL <identifier>
CALL <identifier> (value1, value2, ...)
```

```
PROCEDURE greet (name: STRING)
    OUTPUT "Hello", name
ENDPROCEDURE
```

```
CALL greet("Jason")
```

This is a procedure named `greet` that takes a parameter `name` and outputs a statement `"Hello <name>"`. The variable `name` contains a string value passed into the procedure. For example, if `name` has the value `"Jason"`, then the procedure will output the statement `"Hello Jason"`.

5.3 Functions

Functions operate in a similar way to procedures, except that in addition they **return a single value** to the point at which they are called. Their definition includes the data type of the value returned.

A function with no parameters is defined as follows:

```
FUNCTION <identifier> RETURNS <data type>
    <statement(s)>
ENDFUNCTION
```

A function with parameters is defined as follows:

```
PROCEDURE <identifier> (<param1>:<datatype>,
                        <param2>:<datatype>...) RETURNS <data type>
    <statement(s)>
ENDPROCEDURE
```

The keyword `RETURN` is used as one of the statements within the body of the function to specify the value to be returned. Normally, this will be the last statement in the function definition. When the `RETURN` statement is executed, the value returned replaces the function call in the expression and the expression is then evaluated.

```
FUNCTION find_max (first: INTEGER, second: INTEGER)
    RETURNS INTEGER
    IF first > second
        THEN
            RETURN first
        ELSE
            RETURN second
    ENDIF
ENDFUNCTION
```

```
OUTPUT "Larger number is", find_max(7, 11)
```

This function named `find_max` will take two parameters `first` and `second`, compares the values of `first` and `second`, and returns the larger value of the two.

5.4 Passed by value and by reference

Unless otherwise stated, it should be assumed that parameters are **passed by value**. That means that its actual value is passed to the subroutine, where it is treated as a local variable. Changing a parameter inside the subroutine will not affect its value outside the subroutine. All parameters are passed by value in Python.

In other programming languages, parameters may be passed by value but they may also be **passed by reference**. In this case, the **address**, and **not the value**, of the parameter is passed to the subroutine.

To specify whether a parameter is passed by value or by reference, the keywords **BYVAL** and **BYREF** precede the parameter in the definition of the procedure. If there are several parameters passed by the same method, the BYVAL or BYREF keyword need not be repeated.

```
PROCEDURE SWAP (BYREF X:INTEGER, Y:INTEGER)
```

```
    DECLARE Temp: INTEGER
```

```
    Temp ← X
```

```
    X ← Y
```

```
    Y ← Temp
```

```
ENDPROCEDURE
```

```
PROCEDURE SHOW (BYVAL X:INTEGER, Y:INTEGER)
```

```
    PRINT "X=", X
```

```
    PRINT "Y=", Y
```

```
ENDPROCEDURE
```

MAIN

```
    DECLARE A: INTEGER
```

```
    DECLARE B: INTEGER
```

```
    A ← 77
```

```
    B ← 11
```

```
    PRINT "Before swap:"
```

```
    SHOW(A,B) // 1st call
```

```
    SWAP(A,B)
```

```
    PRINT "After swap:"
```

```
    SHOW(A,B) // 2nd call
```

```
ENDMAIN
```

SWAP [By reference]		
Address		value
BEC3	X	
BED4	Y	
CAB6	Temp	

SHOW [By value] 1 ST call		
Address		value
E432	X	
E420	Y	

SHOW [By value] 2 ND call		
Address		value
E768	X	
E790	Y	

MAIN		
Address		value
33BA	A	
33CD	B	

Output of program:

Before swap:

X =

Y =

After swap:

X =

Y =

When parameters are passed by reference, a reference (address) to that variable is passed to the procedure when it is called. If that value is changed in the procedure, this change is reflected in the variable which was passed into it, after the procedure has terminated.

6. Decision Table

A decision table is a precise way of modelling logic. Some programs require multiple decisions to produce the correct output. Managing all possible outcomes of multiple decisions can be a difficult task. A decision table can help to organise the possible decision outcome combinations. Each possible combination of conditions is considered in turn and the action required.

6.1 Types of decision tables

- Limited Entry Decision Table (Limited Condition/Limited Action)
- Mixed Entry Decision Table (Limited Condition/Extended Action)
- Extended Entry Decision Table (Extended Condition/Extended Action)

Limited condition/action means that the conditions/actions are binary (Yes/No).

Extended condition/action means that the conditions/actions can take several values.

Example (Credit Worthy Customer)

A clerk, in assessing the amount of discount allowed on a customer's order is required to comply with the following policy: "Any order of \$500 or more received from credit worthy customer attracts discount of 5%, and, similarly, orders of less than \$500 attract discount of 3%. Other circumstances must be referred to the supervisor for a decision."

Solution using Limited Entry Decision Table

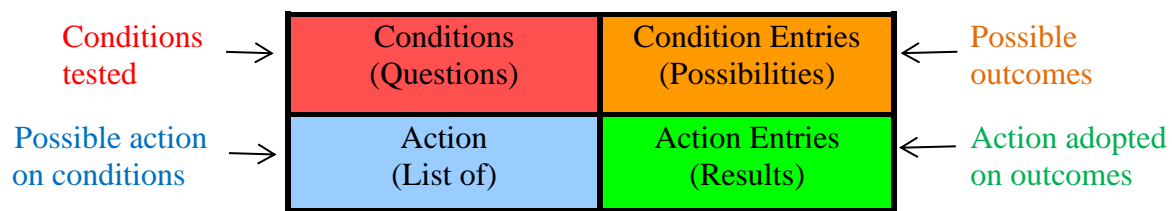
CONDITIONS	OUTCOMES			
	1	2	3	4
Is order \$500 or more?	Y	Y	N	N
Is credit worthy?	Y	N	Y	N
ACTIONS				
Allow discount 3%			X	
Allow discount 5%	X			
Refer to supervisor		X		X

Solution using Mixed Entry Decision Table

CONDITIONS	OUTCOMES			
	1	2	3	4
Is order \$500 or more?	Y	Y	N	N
Is credit worthy?	Y	N	Y	N
ACTIONS				
Allow discount of	5%		3%	
Refer to supervisor		X		X

6.2 Basic format of a decision table

The table can be considered as being divided up into four main quadrants.



In the top left quadrant, list all possible conditions or questions which are to be involved in the decision-making process.

In the bottom left quadrant, list the actions that can be taken.

In the top right quadrant, **Y** means true and **N** means false, which indicates the conditions being applied in a particular column. The rules which are given the numbers 1, 2, 3, etc, is determined by the vertical column underneath each of the numbers. Use dash '—' to mean that the condition is insignificant or impossible.

Finally, in the bottom-right quadrant, list the actions selected or the results, which relate to the rules indicated in the columns above. For limited action entries, use 'X' to **denote a particular action has been chosen**, or blank to ignore. For extended action entries, any statements may be used.

6.3 Decision table rules (number of possible outcomes)

To every decision, there are usually 2 possible outcomes. Therefore for n decisions, there are 2^n possible outcomes. However, usually several possibilities may result from these related decision.

6.4 Advantages & Disadvantages of using decision table

The cause and effect relationship is shown clearly by each column and it is easy to trace from conditions to actions and vice versa. Since there is a maximum number of combinations, it is possible to anticipate and check all combinations (2^n). Columns can be grouped to facilitate analysis which provides concise descriptions of logical complex situations. These make it easier for testing of all outcomes.

A decision table is easier to draw and change than a flowchart and it is much more convenient to use when a large number of logical alternatives exist.

However, large decision tables can become incomprehensible and difficult to modify, especially when there are too many conditions which lead to many more columns. For example, 4 conditions could mean $2^4 = 16$ columns, 5 conditions could mean $2^5 = 32$ columns.

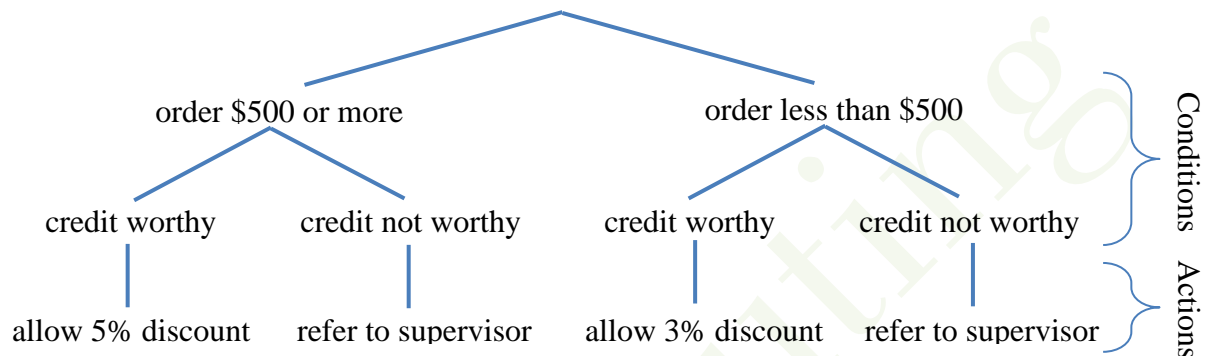
Flowcharts are better to express the total sequence of events needed to solve a problem from start to the end. Flowcharts are also more familiar to many programmers and beginners.

6.5 Decision Tree

A decision tree shows the possible choices to take in reaching an outcome. It is identical to probability tree and is extremely easy to use. Working from the root, we can easily see which condition satisfies any particular situation and then follow the appropriate branch of the tree. Eventually, after all the appropriate conditions have been fulfilled, we will end up at the leaf of the tree where the action to be taken is encountered.

Example (Credit Worthy Customer)

Solution using decision tree



6.6 Removing Redundancies

A limited entry decision table can be simplified by removing redundancies. To find redundancies, look at all similar actions and then check whether the conditions are required. If the conditions are insignificant, they can be replaced with a dash “—”.

If the resultant actions are impossible, they can be denoted with an asterisk during analysis and the rule column be omitted after removing redundancies.

After simplifying, we need to do a completeness check to ensure that all the possible conditions result in the correct actions.

Example (Go to work)

You have to decide how to go to work. You only go to work on a weekday. If the time is before 7am, you will take breakfast and walk to work. If the time is between 7am and 8am, you take breakfast and go to work by bus. If the time is after 8am, you skip breakfast and go to work by bus.

Solution using Limited Entry Decision Table

CONDITIONS	OUTCOMES							
	1	2	3	4	5	6	7	8
Is it a weekday?	Y	Y	Y	Y	N	N	N	N
Is it before 7am?	Y	Y	N	N	Y	Y	N	N
Is it after 8am?	Y	N	Y	N	Y	N	Y	N
ACTIONS								
Breakfast	*	X		X	*			
Walk to work	*	X			*			
Go by bus	*		X	X	*			

(*no such outcomes exist and omit these columns)

Remove redundancies

Look at each action and then check whether the conditions are required.

In column 2, **take breakfast** and **walk to work** only applies if conditions 1 and 2 are true, and condition 3 is false. There is no redundancy here.

In column 3, **go by bus** only applies if conditions 1 and 3 are true, and condition 2 is false. There is also no redundancy here.

In column 4, **take breakfast** and **go by bus** only applies if condition 1 is true, and conditions 2 and 3 are false. Hence no redundancy.

For columns 6 to 8, none of the actions need to be taken for each of these columns. Therefore simplify the table by putting a dash in the cells where the condition is insignificant (can be true or false) – the action will be the same.

CONDITIONS	OUTCOMES			
	1	2	3	4
Is it a weekday?	Y	Y	Y	N
Is it before 7am?	Y	N	N	-
Is it after 8am?	N	Y	N	-
ACTIONS				
Breakfast				
Walk to work				
Go by bus				

Complete the decision table by putting an 'X' for the action taken

Example 6.3 (Produce toys)

A company produces toys. It accepts a product, if it passes the following three tests.

- All dimensions are correct
- Safety tests are passed
- Paint tests are passed

If the first test is passed but exactly one of the other two fails, the toy is sent for repair. Otherwise the toy is rejected.

- Create a decision table showing all the possible outcomes and results.
- Simplify your solution by removing redundancies.

(a)

CONDITIONS	OUTCOMES							
	1	2	3	4	5	6	7	8
All dimensions are correct?	Y	Y	Y	Y	N	N	N	N
Safety tests are passed?	Y	Y	N	N	Y	Y	N	N
Paint tests are passed?	Y	N	Y	N	Y	N	Y	N
ACTIONS								
Accept product	X							
Repair product		X	X					
Reject product				X	X	X	X	X

(b)

CONDITIONS	OUTCOMES				
	1	2	3	4	5
All dimensions are correct?					
Safety tests are passed?					
Paint tests are passed?					
ACTIONS					
Accept product	X				
Repair product		X	X		
Reject product				X	X

Complete the decision table by putting 'Y' or 'N' to indicate condition as yes or no

7. Reference

Pseudocode guide for teachers: Cambridge International AS & A level Computer Science 9618. Cambridge University Press & Assessment.

- <https://www.cambridgeinternational.org/Images/697401-2026-pseudocode-guide-for-teachers.pdf>