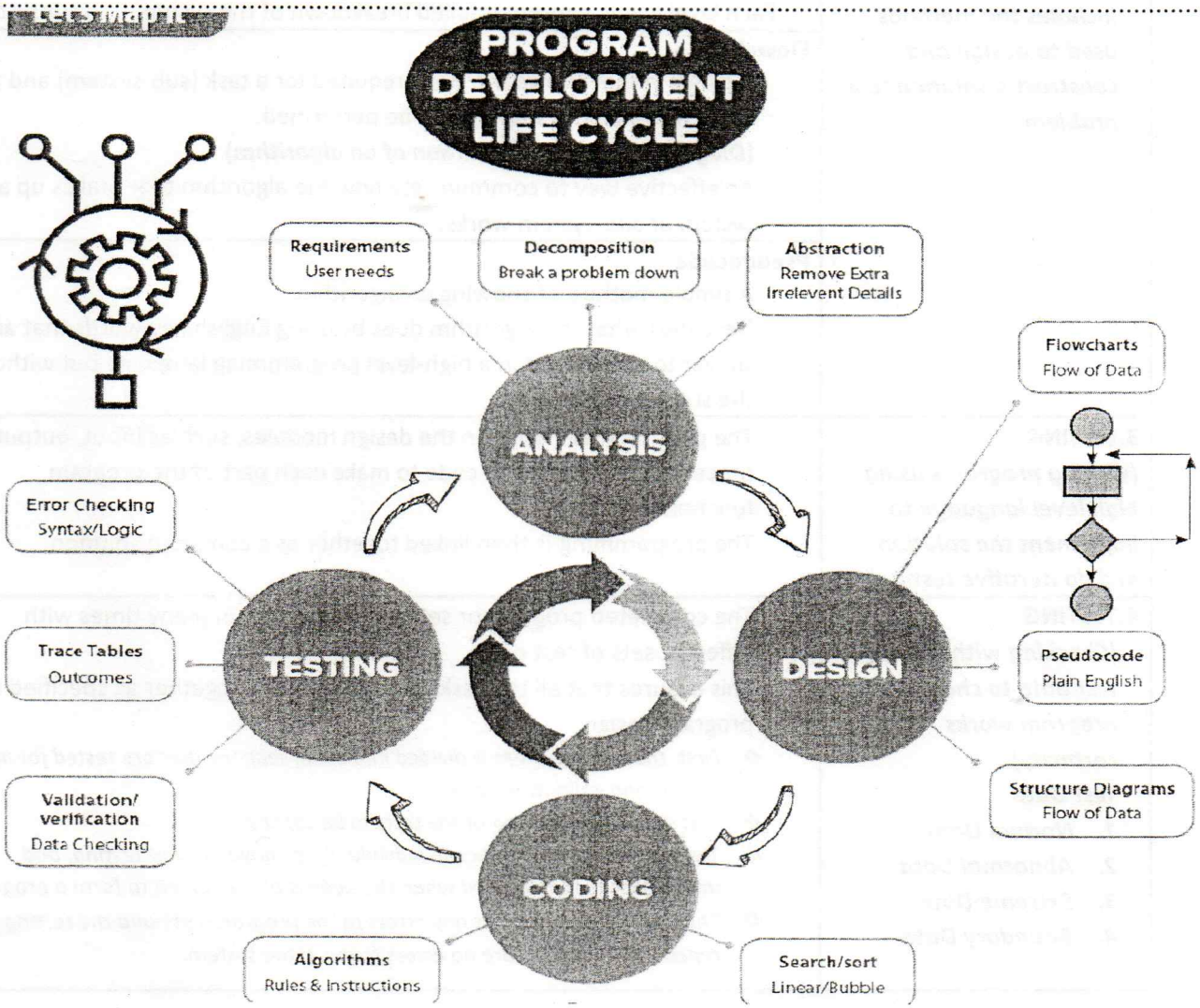
ALGORITHM, DESIGN & PROBLEM SOLVING

UNIT: 7

Topic Objectives:

- Understand the program development life cycle, limited to: analysis, design, coding and testing
- Understand that every computer system is made up of subsystems, which are made up of further sub-systems
- Understand how a problem can be decomposed into its component parts
- Use different methods to design and construct a solution to a problem
- Use different methods to design and construct a solution to a problem
- Understand the need for validation checks to be made on input data and the different types of validation check
- Understand the need for verification checks to be made on input data and the different types of verification check
- Suggest and apply suitable test data



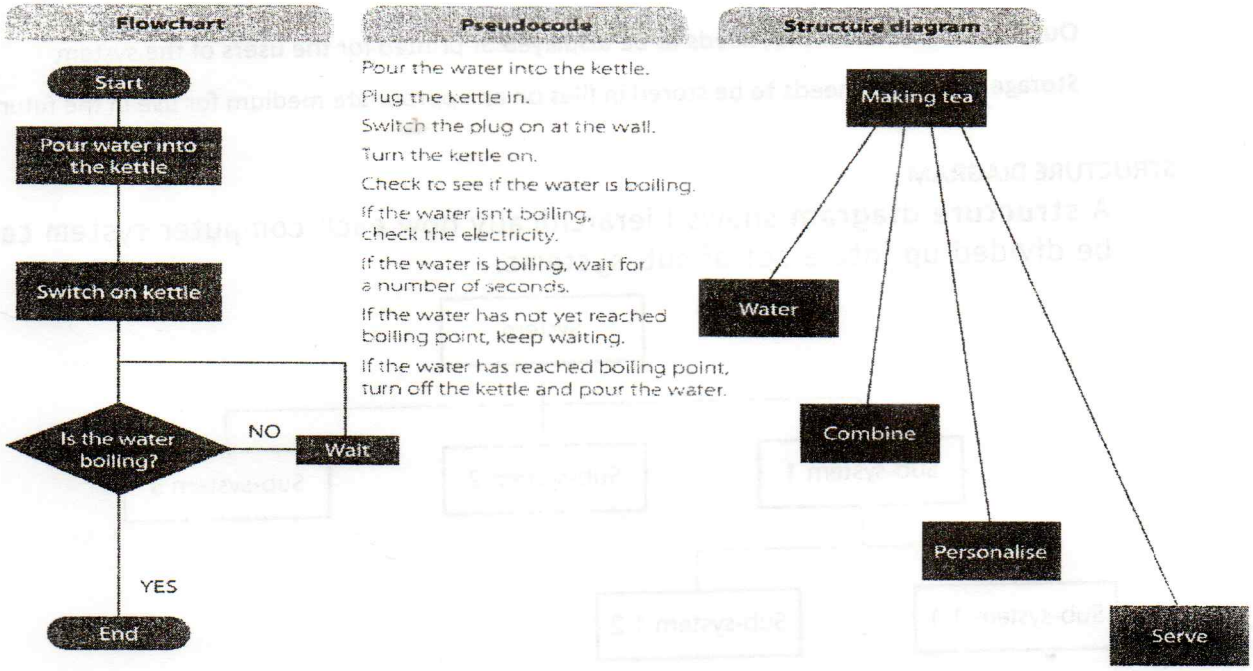
PROGRAM DEVELOPMENT LIFE CYCLE (Stages in program development)	
1.ANALYSIS *(Identifying and understanding the requirements to solve the problem)*	**Abstraction** The process of selecting the information in a problem and focusing only on the important/key parts of that information. During abstraction, any detail that is irrelevant to the problem is ignored.
	Decomposition The process of breaking down a complex problem into small manageable pieces. A problem becomes a series of small tasks that can be handled first individually and then collectively as part of the bigger program.
	Identification of the problem & requirements
2.DESIGN *(Using the program specification from the analysis stage to show how the program should be developed)* *Includes the methods used to design and construct a solution to a problem*	**Decomposition** The process of breaking down a complex problem into small manageable pieces. Identifying the **input, process, output and storage** required
	Structure Diagram A diagram that shows the design of a computer system in a hierarchical way. Can be used to show top-down design in a diagrammatic form Each level giving a more detailed breakdown of the system into subsystems.
	Flowchart A diagram that shows the steps required for a task (sub-system) and the order in which the steps are to be performed. (***Diagrammatic representation of an algorithm***) An effective way to communicate how the algorithm that makes up a system or sub-system works.
	Pseudocode A simple method of showing an algorithm; Describes what the algorithm does by using English key words that are very similar to those used in a high-level programming language but without the strict syntax rules.
3.CODING *(Writing programs using high level language to implement the solution and do iterative testing)*	The programmers work on the design modules, such as input, output and processes, and write the code to make each part of the program functional. The programming is then linked together as a complete solution.
4.TESTING *(Checking with sample test data to check if the program works correctly)* *Test Data* 1. *Normal Data* 2. *Abnormal Data* 3. *Extreme Data* 4. *Boundary Data*	The completed program or set of programs is run many times with different sets of test data. This ensures that all the tasks completed work together as specified in the program design. ❖ *First, the entire system is divided into small sections that are tested for input, output and validation rules.* ❖ *Test data is used for all of the tests to be carried out.* ❖ *Testing each part of the code individually is known as unit testing, and integration testing is used when this code is all combined to form a program.* ❖ *The testing team reports any errors to the programmers and the testing is repeated until there are no errors found in the system.*

EXAMPLE:

Problem: A person wishes to make a cup of tea.

1. **ANALYSIS**
   a. **Abstraction**
      - **Information obtained**: teabags, milk, sugar, white mug, water, kettle, teapot.
      - **Abstracted information**: can ignore the colour of the mug as this is not relevant to this particular problem.
   b. **Decomposition:**
      This complex problem can be broken down into four smaller problems: water, combine, personalise and serve.
      Decomposed problem:
      Water: Fill the kettle with water and heat it until it reaches boiling point.
      Combine: Add the water to a teabag in either a teapot or mug and leave for a set time.
      Personalise: Remove tea bag from water, pour tea into a mug, and add the sugar and milk to taste.
      Serve: Stir the tea and drink it from the mug.
   c. **Identifying the problem & the requirements:**

2. DESIGN



(NOTE: Structure Diagram must be drawn in hierarchical form. Coding and testing are not given as it's a real time task to be done. This example is just to make the learners understand the stages)

SYSTEMS & SUB-SYSTEMS

Top-down design is the decomposition of a computer system into a set of subsystems, then breaking each sub-system down into a set of smaller sub-systems, until each sub-system just performs a single action.

The process of breaking down into smaller sub-systems is called stepwise refinement.
This is an effective way of designing a computer system to provide a solution to a problem

- Since each part of the problem is broken down into smaller more manageable problems.
- Works for the development of both large and small computer systems.
- When larger computer systems are being developed this means that several programmers can work independently to develop and test different sub-systems for the same system at the same time.
- Reduces the development and testing time.

DECOMPOSING A PROBLEM

Any problem that uses a computer system for its solution needs to be decomposed into its component parts. The component parts of any computer system are:

Inputs - the data used by the system that needs to be entered while the system is active

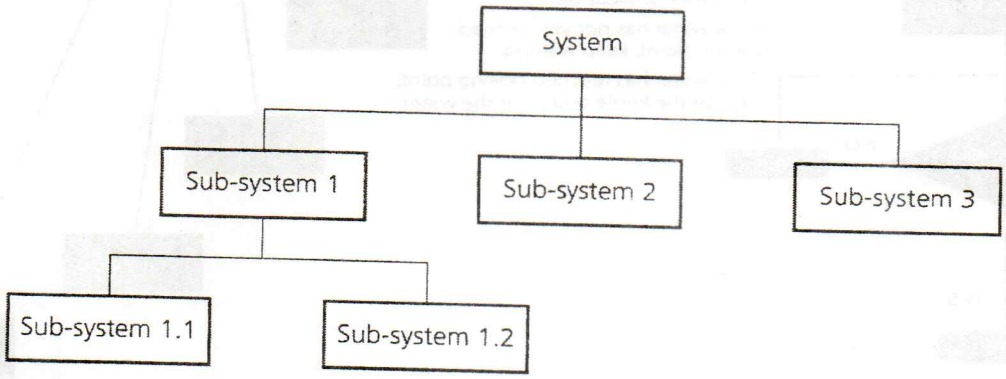
Processes - the tasks that need to be performed using the input data and any other previously stored data

Outputs - information that needs to be displayed or printed for the users of the system

Storage - data that needs to be stored in files on an appropriate medium for use in the future.

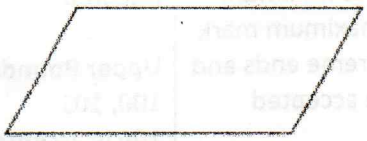
STRUCTURE DIAGRAM

A **structure diagram** shows hierarchically how each computer system can be divided up into a set of sub-systems.



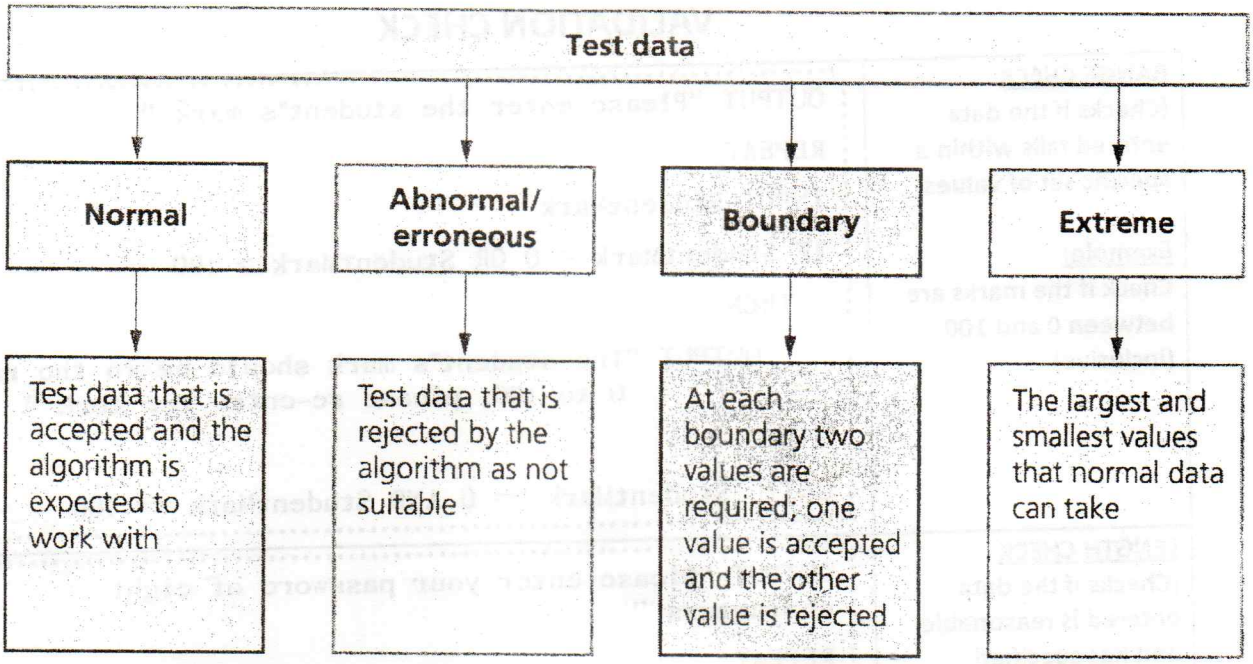
FLOWCHART

Flowchart symbols

	**Flow line**	An arrow represents control passing between the connected shapes.
	**Process**	This shape represents something being performed or done.
	**Subroutine**	This shape represents a subroutine call that will relate to a separate, non-linked flowchart.
	**Input/Output**	This shape represents the input or output of something into or out of the flowchart.
	**Decision**	This shape represents a decision (Yes/No or True/False) that results in two lines representing the different possible outcomes.
	**Terminator**	This shape represents the 'Start' and 'Stop' of the process.

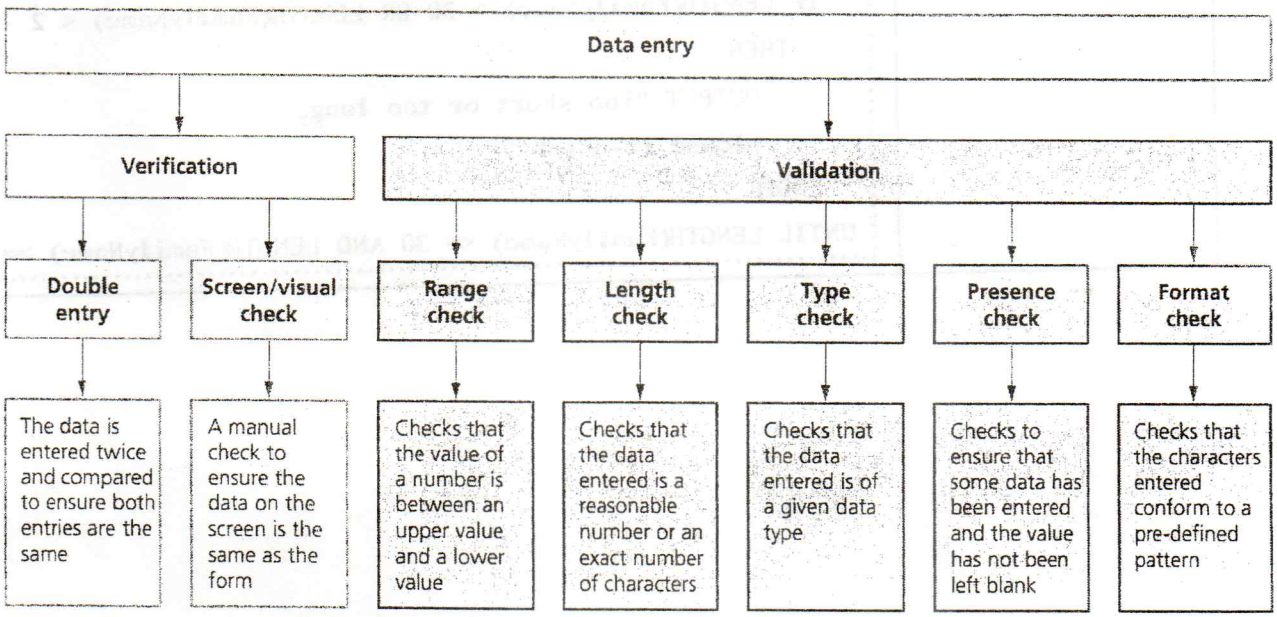
TEST DATA - A set of test data is all the items of data required to work through a solution.

NORMAL DATA	ABNORMAL/ ERRONEOUS DATA	EXTREME DATA	BOUNDARY DATA
Data that is always accepted and falls within the criteria specified	**Data that is always rejected and falls outside the criteria specified**	**The largest and the smallest acceptable value**	**The largest/ smallest acceptable value and the corresponding largest/ smallest rejected value**
Example 1: To check the mark of a student (out of 100)			
73 It is a normal data as it is within the limit of acceptability and should be always accepted	**-5, fifty, 110 etc.** These are abnormal data as: -5 is a negative number Fifty is written in words 110 is above 100 **All these values should be rejected**	**Lower Extreme data:0** **Upper Extreme:100** Both these values indicate the minimum and the maximum mark at the extreme ends and should be accepted	**Lower Boundary Values** **0, -1** **0 is accepted and -1 is rejected** **Upper Boundary Values** **100, 101** **100 is accepted** **101 is rejected**
Example 2: To check the if the weight of the student is between 40.50 and 55.00 (inclusive)			
43.25 It is a normal data as it is within the limit of acceptability and should be always accepted	**-5, fifty, 60.00 etc.** These are abnormal data as: -5 is a negative number Fifty is written in words 60.00 is above 55.00 **All these values should be rejected**	**Lower Extreme:40.50** **Upper Extreme:55.00** Both these values indicate the minimum and the maximum weight at the extreme ends and should be accepted	**Lower Boundary Values** **40.50, 40.49** **40.50 is accepted and -40.49 is rejected** **Upper Boundary Values** **55.00, 55.01** **55.00 is accepted** **55.01 is rejected**



VALIDATION & VERIFICATION

VALIDATION	VERIFICATION
Validation checks whether data to be entered is possible/sensible // computer check	Verification checks that data entered is the data that was intended to be entered // can be a human check // matches the source
Validation is the automated checking by a program that data is reasonable before it is accepted into a computer system.	
Includes Range Check, Length Check, Type Check, Presence Check, Format Check, Check digit	Includes double entry and visual/ screen check



VALIDATION CHECK

RANGE CHECK
(Checks if the data entered falls within a specific set of values)

Example:
Check if the marks are between 0 and 100 (inclusive)

```
OUTPUT "Please enter the student's mark "
REPEAT
INPUT StudentMark
IF StudentMark < 0 OR StudentMark > 100
  THEN
    OUTPUT "The student's mark should be in the range
            0 to 100, please re-enter the mark "
ENDIF
UNTIL StudentMark >= 0 AND StudentMark <= 100
```

LENGTH CHECK
(Checks if the data entered is reasonable and has specified number of characters)

Example 1:
Check if the password has exactly 8 characters

```
OUTPUT "Please enter your password of eight
characters "
REPEAT
    INPUT Password
    IF LENGTH(Password) <> 8
      THEN
        OUTPUT "Your password must be exactly eight
                characters, please re-enter "
    ENDIF
UNTIL LENGTH(Password) = 8
```

Example 2:
A family name could be between two and thirty characters inclusive so that names with one character or thirty-one or more characters would be rejected

```
OUTPUT "Please enter your family name "
REPEAT
  INPUT FamilyName
  IF LENGTH(FamilyName) > 30 OR LENGTH(FamilyName) < 2
    THEN
      OUTPUT "Too short or too long,
      please re-enter "
  ENDIF
UNTIL LENGTH(FamilyName) <= 30 AND LENGTH(FamilyName) >= 2
```

TYPE CHECK
Checks that the data entered is of a given data type

Example:
Checks that the number of brothers or sisters would be an integer (whole number)

```
OUTPUT "How many brothers do you have? "
REPEAT
INPUT NumberOfBrothers
IF NumberOfBrothers <> DIV(NumberOfBrothers, 1)
  THEN
    OUTPUT "This must be a whole number, please re-enter"
ENDIF
UNTIL NumberOfBrothers = DIV(NumberOfBrothers, 1)
```

PRESENCE CHECK
Checks to ensure that some data has been entered and the value has not been left blank.

Example:
An email address for an online transaction must be completed.

```
OUTPUT "Please enter your email address "
REPEAT
INPUT EmailAddress
IF EmailAddress = ""
  THEN
    OUTPUT "*=Required "
ENDIF
UNTIL EmailAddress <> ""
```

FORMAT CHECK
Checks that the characters entered conform to a pre-defined pattern.

Example:
The ID number must in the form NCC where N is a number and C can be any character.

The date is given in the format DD/MM/YYYY

The cub number must be in the form CUB9999.

Example 1: An ID number entered needs to be 1 number followed by 2 characters.
Ex:5AE
This algorithm checks the characters and outputs whether it meets the required format.

```
1. INPUT IdNumber
   //check if the first character is a number, and characters 2 and 3 are strings
2. IF(SUBSTRING(IdNumber, 0, 1) .IsNumeric = TRUE AND
   SUBSTRING(IdNumber, 1, 2) = TRUE THEN
3.     OUTPUT ("Valid")
4. ELSE
5.     OUTPUT ("Invalid" )
6. ENDIF
```

Note that SUBSTRING (String, X, Y) starts at character x in the string, and returns y number of characters

Example 2
Using a loop:
This algorithm checks whether an input is in the format two numbers,/, two numbers,/, four numbers. Example:12/12/1986

```
INPUT Date
//loop while the data is not in the correct format
WHILE (SUBSTRING(Date, 0, 2).IsNumeric = FALSE OR //2 numbers
       SUBSTRING(Date, 2, 1) <> "/" OR // 1st slash
       SUBSTRING(Date, 3, 2).IsNumeric = FALSE OR //2 numbers
       SUBSTRING(Date, 5, 1) <> "/" OR //2nd slash
       SUBSTRING(Date, 6, 4).IsNumeric = FALSE) DO // 4 numbers, year
   OUTPUT("Invalid")
   INPUT Date
ENDWHILE
OUTPUT("Valid")
```

CHECKDIGIT

A **check digit** is calculated from a set of numbers, and is input with the numbers.

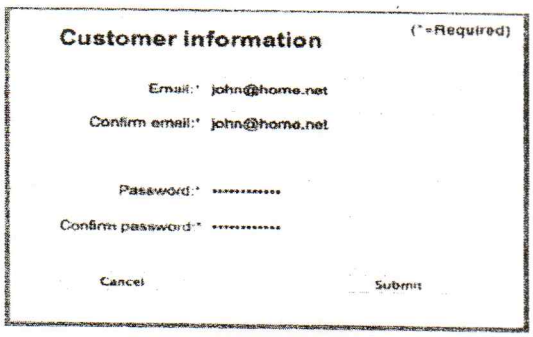
When the data has been input, the calculation is performed on the data again and the result is compared with the final number (the check digit).

Example: This algorithm calculates the check digit using the method 5 digit number input. 9999, 4 digits used to calculate 5th check digit, and compares it to the input value.

```
INPUT Code
//extract 1st digit
Digit1 ← Code DIV 1000
Code ← Code - (1000 * Digit1)
//extract 2nd digit
digit2 ← Code DIV 100
Code ← Code - (100 * Digit1)
//extract 3rd digit
Digit3 ← Code DIV 10
Code ← Code - (10 * Digit1)
//extract 4th digit
Digit4 ← DIV(Code, 1)
CheckDigit ← Code
//calculate check digit from data entered
//multiply by position and add together
Total ← (Digit1 * 4) + (Digit2 * 3) + (Digit2 * 2) + Digit3

NewCheckDigit ← MOD(Total, 11) //find Mod 11 of total
NewCheckDigit ← 11 - NewCheckDigit //subtract result from newCheckDigit
//check if the calculated check digit is the same as the last digit in the //code
IF NewCheckDigit = CheckDigit THEN
   OUTPUT("Correct data entry")
ELSE
   OUTPUT("Incorrect data entry")
ENDIF
```

VERIFICATION

DOUBLE ENTRY	SCREEN/VISUAL CHECK
1. The data is entered twice, sometimes by different operators. 2. The computer system compares both entries 3. If they are different outputs an error message requesting that the data to be entered again.  	❖ It is a manual check completed by the user who is entering the data. ❖ When the data entry is complete: 1. The data is displayed on the screen 2. The user is asked to confirm that it is correct before continuing. 3. The user either checks the data on the screen against a paper document that is being used as an input form or, confirms whether it is correct from their own knowledge.