7 Algorithm Design & Problem Solving

The Program Development Life Cycle

1. **Analysis:** Identifying and understanding the requirements to solve the problem
    - **Abstraction**:
        - Abstraction is the process of removing unnecessary details and focusing only on the important information needed to solve a problem.
        - This makes the problem easier to understand and solve.

    - **Decomposition of the problem**:
        - The process of breaking down a complex problem into smaller manageable parts.
        - Each sub-problem can then be solved separately.
        - This makes the overall problem easier to understand and solve.

    - **Identification of the problem and requirements**

2. **Design**
    - Using the program specification from the analysis stage to show how the program should be developed.
    - This provides a clear plan for how the program will work.
    - **Structure diagrams**, **flowcharts** & **pseudocode** can be used to present the design of a solution to a problem.

3. **Coding**
    - Writing the program code in a programming language
    - Use pseudocode, flowcharts & structure diagrams from design stage to create code
    - Testing the code in small parts as you write it allow iterative testing
    - Documenting your codes with comments to make easier to maintain

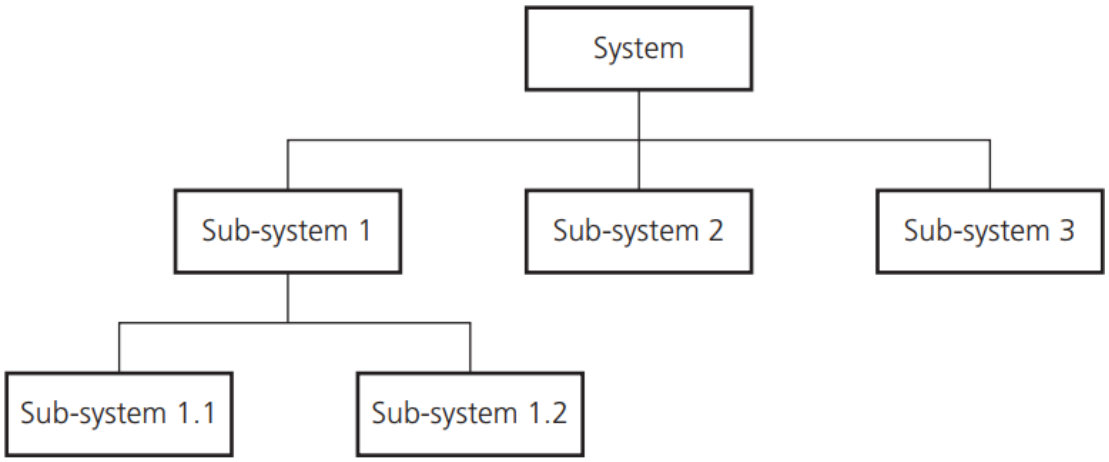
4. **Testing**
    - The final stage of testing the whole program for errors.
    - Use the test data to ensure the whole program works correctly
    - Errors found are identified and corrected.

Computer Systems

- Every **computer system** is made up of **sub-systems**, which are made up of **further sub-systems** which each have a single purpose (this decomposition is known as **top-down design**)
- The **component parts** of decomposing a system
    - **Inputs**: the data used by the system that needs to be entered
    - **Processes**: the tasks that need to be performed using the input & stored data
    - **Output**: the data that is produced by the system
    - **Storage**: the data that is stored on a physical device e.g. hard drive

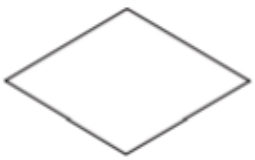
➔ **Structure diagram**

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



➔ **Flowchart**

- A **flowchart** shows diagrammatically the steps required to complete a task and the order that they are to be performed.

	**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.

➔ **Pseudocode**

A method of describing an algorithm using English keywords. This is a simple method of showing an algorithm

Validation and Verification

Validation

Validation is an automatic check carried by the computer in order to make sure that input data is reasonable before it is accepted into a computer system

Range Check
*Checks that the value of a **number** is **between an upper value and a lower value*** *e.g. percentage mark between 0-100*

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

Length Check	
*Checks that **data** contains an **exact number of characters*** *e.g. password must be exactly 8 characters*	

OUTPUT "Please enter password of 8 characters"
REPEAT
  INPUT Password
  IF LENGTH(Password) <> 8
  THEN
  OUTPUT "The password must be exactly 8 characters, please re-enter"
  ENDIF
UNTIL LENGTH(Password) = 8
	OUTPUT "Please enter family name"
	REPEAT
	  INPUT Name
	  IF LENGTH(Name) > 30 OR LENGTH(Name) < 2
	  THEN
	  OUTPUT "Too short/long, please re-enter"
	  ENDIF
	UNTIL LENGTH(Name) <= 30 OR LENGTH(Name) >= 2

Type Check	***Presence Check***
*Checks that the **data** entered is of a **correct data type*** *e.g. # of people must be an integer*	*Checks that **some data has been entered*** *e.g. email address must be filled in*

OUTPUT "Enter number of people"
REPEAT
  INPUT People
  IF People <> ROUND(People, 0)
  THEN
  OUTPUT "Must be a whole number, please re-enter"
  ENDIF
UNTIL People = ROUND(People, 0)
	OUTPUT "Enter email address*"
	REPEAT
	  INPUT Email
	  IF Email = ""
	  THEN
	  OUTPUT "*=Required"
	  ENDIF
	UNTIL Email <> ""

Format Check & Check Digit	
Format Check	***Check Digit***
*Checks that the entered data is in **correct format*** *e.g. ID Number (A156999)*	*An **additional digit** attached to a data,* ***calculated from the other digits*** *e.g. barcodes, product codes, ISBN, VIN*

INPUT ID
 	
FirstCharacter ← SUBSTRING(ID, 1, 1)
 	
IF FirstCharacter = 'A'
THEN
OUTPUT "Accepted"
ELSE
OUTPUT "Invalid"
ENDIF
	Check digits are used to identify errors in data entry caused by mistyping or mis-scanning a barcode:
	- An incorrect digit entered
	- Swapped digits
	- Missing digit
	- Extra digit
	- Mispronunciation of digits

Verification

Verification checks that no changes are made to the data on entry, data has been accurately copied from one source to another

Double Entry	***Visual Check***
The data is entered twice and compared. If they don't match, an error message is shown	A message asks the user to confirm the data is correct before proceeding. If it isn't, the user then enters the data again

Test Data

Test data checks that a program works as expected, accepts reasonable data, & rejects invalid data

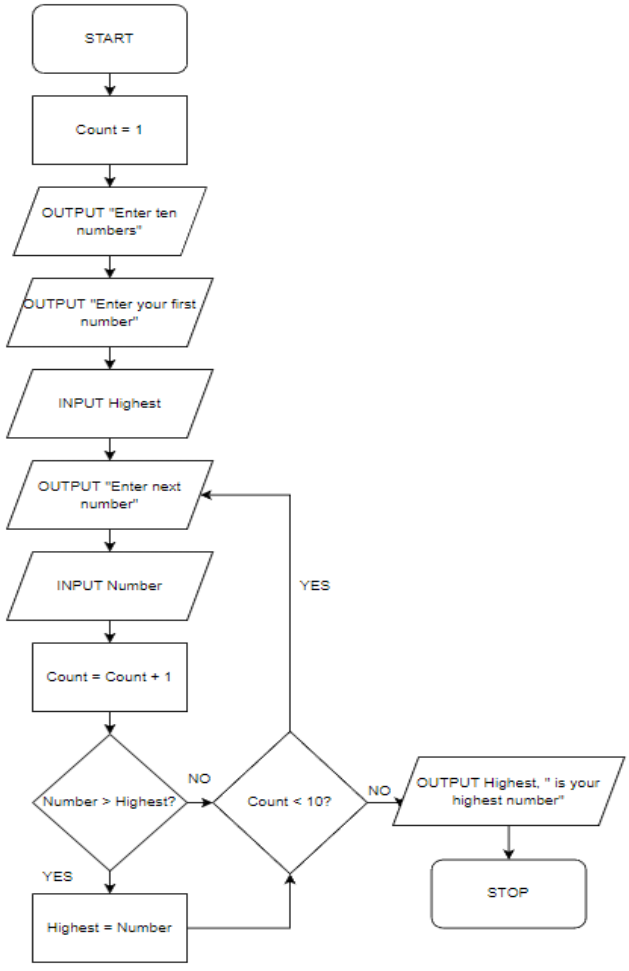
Normal data	**Abnormal data**
Test data that is accepted by a program	Test data that is rejected by a program
Eg: To check the mark of a student out of 100. **Normal data:** 1, 5, 73, 85 Any values that would be within the limit of acceptability	**Eg: To check the mark of a student out of 100.** **Abnormal data:** -5, fifty, 110 Any values that would be outside of the limit. These data will be rejected.

Extreme data	Boundary data
The largest/smallest acceptable values	The largest/smallest data value acceptable & the corresponding smallest/largest rejected value
Eg: To check the mark of a student out of 100. **Lower extreme values:** 0 0 is accepted **Upper extreme values:** 100 100 is accepted	**Eg: To check the mark of a student out of 100.** **Lower boundary values:** -1, 0 0 is accepted and -1 is rejected **Upper boundary values:** 100, 101 100 is accepted and 101 is rejected

Trace Tables, Flowcharts, Pseudocode

A **trace table** is used to record the results from each step in an algorithm (records the value of a variable each time it changes)
A **dry run** is the manual exercise of working through an algorithm step by step

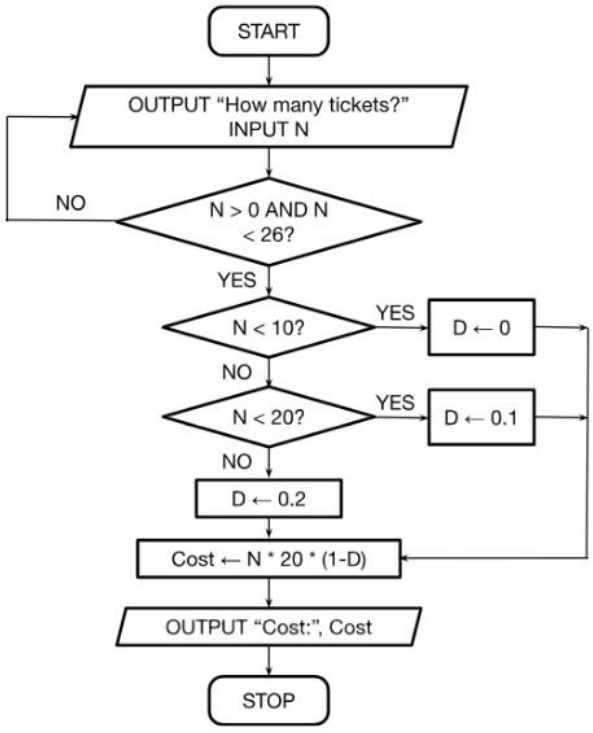
- Determines the highest number of 10 user entered numbers
- The algorithm prompts the user to enter the first number which automatically becomes the highest number entered
- The user is then prompted to enter nine more numbers. If a new number is higher than an older number then it is replaced
- Once all ten numbers are entered, the algorithm outputs which number was the highest
- Example test data to be used is: 4, 3, 7, 1, 8, 3, 6, 9, 12, 10



Trace table for Figure 1: Highest number			
Count	Highest	Number	Output
1			Enter ten numbers
	4		Enter your first number
2		3	Enter your next number
3	7	7	
4		1	
5	8	8	
6		3	
7		6	
8	9	9	
9	12	12	
10		10	12 is your highest number

Tickets are sold at $20 each. If 10 tickets are bought, the discount is 10%, if 20 tickets are bought, the discount is 20%. No more than 25 tickets can be bought at once.

Flowchart



Pseudocode

```
REPEAT
  OUTPUT "How many tickets would you like to buy? "
  INPUT NumberOfTickets
UNTIL NumberOfTickets > 0 AND NumberOfTickets < 26
IF NumberOfTickets < 10
  THEN
    Discount ← 0
  ELSE
    IF NumberOfTickets < 20
      THEN
        Discount ← 0.1
      ELSE
        Discount ← 0.2
    ENDIF
ENDIF
Cost ← NumberOfTickets * 20 * (1 - Discount)
PRINT "Your tickets cost ", Cost
```