Sec 2.1 Problem Solving & Flowcharts

INTRODUCTION:

In order to build a computer system that performs a specific task or solves a given problem, the task or problem has to be clearly defined, showing what is going to be computed and how it is going to be computed. This chapter introduces the tools and techniques that are used to design a software solution that together with the associated computer hardware will form a computer system.

What is a computer system?

A **COMPUTER SYSTEM** is made up of software, data, hardware, communications and people; each computer system can be divided up into a set of sub-systems. Each subsystem can be further divided into sub-systems and so on until each sub-system just performs a single action.

Computer systems can be very large or very small or any size in between; most people interact with many different computer systems during their daily life without realizing it.

For example, when I wake up in the morning I use an app on my smart phone for my alarm, I then check the weather forecast on my computer before I drive to work. The alarm program is a very small computer system; when I check the weather forecast I obtain information from one of the largest computer systems in the world.

STEPS IN PROBLEM SOLVING:

Using computer to solve problem involves far more than just writing a program. In fact many computer applications are carried out without writing new programs. Existing software is used if possible.
Solving a problem may be seen in three main stages:

1. Defining exactly what the problem is.
2. Designing the solution to deal with it.
3. Putting that solution into practice.

In solving an information processing problem we should define:

1. The data which is to be input.
2. The data which is to be output.
3. The data which is to be stored as files.
4. The processing tasks to be carried out.

SYSTEM

A **system** is a set of components forming an integrated whole, which works as a unit. An **information processing system** consists of everything required to carry out a particular processing task.

Tools and techniques:

In order to understand how a computer system is built up and how it works, it is often divided up into sub-systems. This division can be shown using top-down design to produce structure diagrams that demonstrate the modular construction of the system.

Each sub-system can be developed by a programmer as sub-routine or an existing library routine may be already available for use. How each sub-routine works can be shown by using flowcharts or pseudocode.

EXAMPLES OF INFORMATION PROCESSING SYSTEM & ITS SUBSYSTEM:

1. The stock control system for a shop. (The system comprises)
   a. The methods of checking on stock levels, of ordering new goods, of recording their delivery and so on.
   b. The means of communication with suppliers.

2. The working system in an office. This would include:
   a. The methods of communication (by letters, emails or telephone)
   b. The methods of storage information (by filing cabinets or CD's or other storage media)
   c. The methods of producing letters (word processors, or copiers etc)

COMPUTERIZED INFORMATION PROCESSING SYSTEM:

In general a computerized information processing system includes:

1. Computer and other hardware.
2. Computer Software
3. Methods of collecting, checking and inputting data
4. Methods of communication
5. Processing operations carried out on data
6. Method of outputting data.

Advantages of designing a solution to a problem by splitting it up into smaller problems (top- down/modular design)

Produce and describe top-down/modular designs using appropriate techniques, including structure diagrams, showing stepwise refinement

Subsystem:

A unit or device that is part of a larger system. For example, a disk subsystem is a part of a computer system. A bus is a part of the computer. A subsystem usually refers to hardware, but it may be used to describe software. However, "module," "subroutine" and "component" are more typically used to describe parts of software.

Library

In computer science, a **library** is a collection of **non-volatile resources** used by computer programs, often to develop software. These may include configuration data, documentation, help data, message templates, **pre-written code** and **subroutines**, classes, values or type specifications.

Library code is organized in such a way that it can be used by multiple programs that have no connection to each other.

Most compiled languages have a standard library although programmers can also create their own custom libraries. Most modern software systems provide libraries that implement the majority of system services

Routine / Program

A set of programming instructions designed to perform a specific limited task.

- Routine, another name for a computer subprogram
    - Subroutine, a routine (program) inside another routine (program)

Subroutine

In computer programming, a **subroutine** is a sequence of program instructions that perform a specific task, packaged as a unit. This unit can then be used in programs wherever that particular task should be performed.

Subprograms may be defined within programs, or separately in **libraries** that can be used by multiple programs. In different programming languages, a subroutine may be called a **procedure**, a **function**, a **routine**, a method, or a **subprogram**. The generic term **callable unit** is sometimes used.

Stepwise refinement:

Stepwise refinement is the process of developing a modular design by splitting a problem into smaller sub-tasks, which themselves are repeatedly split into even smaller sub-tasks until each is just one element of the final program.

Top down/modular design:

A Top-down design is when a problem is split into smaller sub-problems, which themselves are split into even smaller sub-problems until each is just one element of the final program.
Top down design provides a method of producing computer programs or systems. These are the main features of top-down design:

- The program is broken down into parts of modules
- The parts or modules are broken down into more parts or modules.
- Soon each part or module is easy to produce because it contains enough detail.

Breaking problems into smaller parts makes it clear what needs to be done. At each stage of refinement, the problem becomes less complex and easier to work out.

Benefits and drawbacks of modular programs

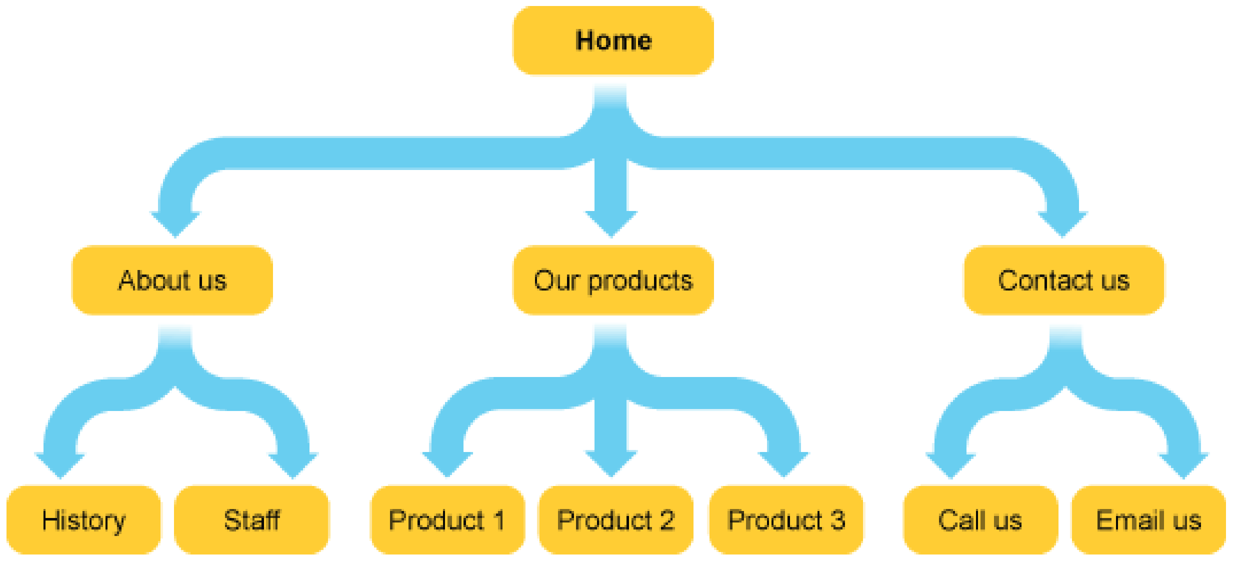
Benefits	Drawbacks
Smaller problems are easier to understand	Modules must be linked and additional testing must be carried out to make sure that the links work correctly.
Smaller problems are easier to test and debug	Programmers must be sure that cross-referencing is done
Development can be shared between a team of programmers- each person programming different modules according to their tasks and skills.	Interface between modules must be planned
Code from previously programmed modules can be reused.	

Data structure diagram

Data structure diagram (**DSD**) is a diagram of the conceptual data model which documents the **entities** and their **relationships**, as well as the **constraints** that connect to them.

The basic graphic notation elements of DSDs are boxes which represent entities. The arrow symbol represents relationships. Data structure diagrams are most useful for documenting complex data entities.

A structure diagram shows the overall structure of a multimedia product.



Structure diagrams

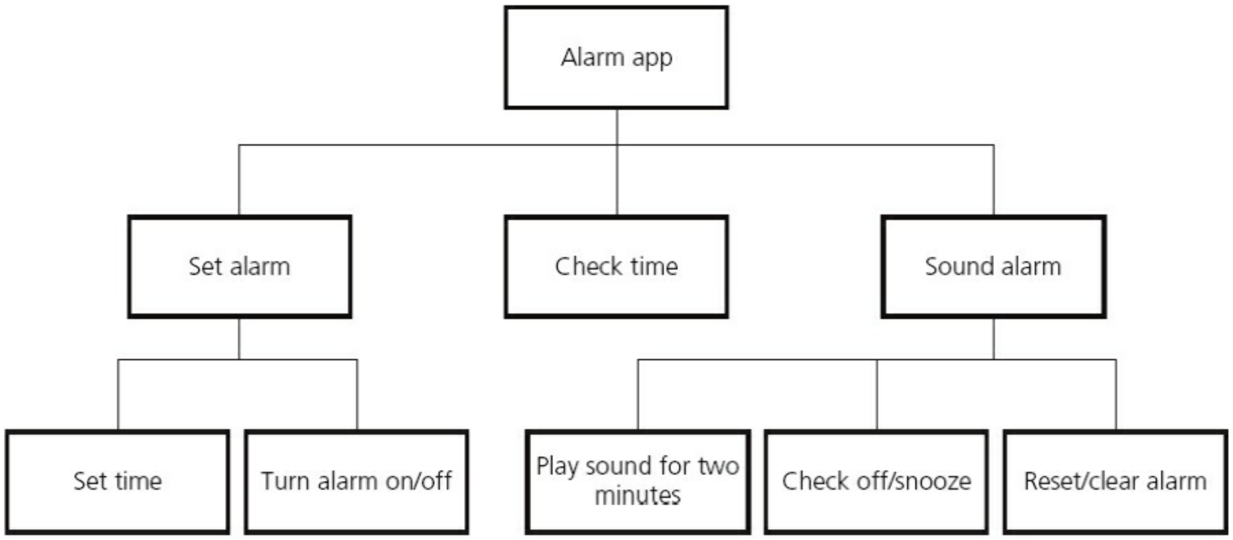
A structure diagram is a pictorial representation of a modular system.

The initial stage in the design of an algorithm to solve a more complex problem is to break down the problem into smaller units that can be considered as separate problems.

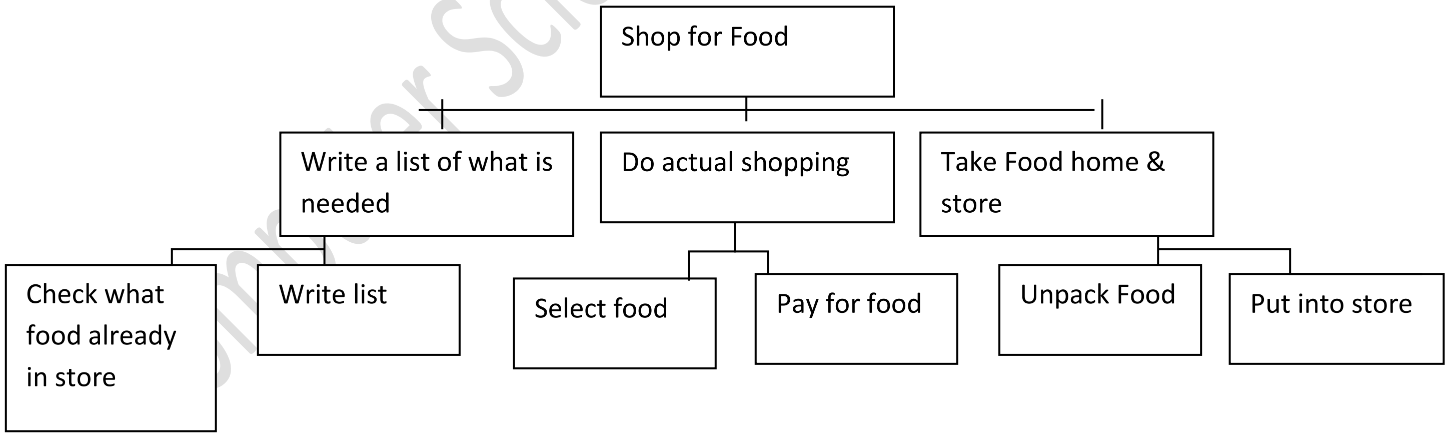
A diagram can be used to show how this is achieved. This is known as structure diagram.

Structure diagrams are particularly useful when the problem has been broken down into these smaller tasks and then broken down into even smaller subtasks. This Method of solving problem is known as **top-down design** or **stepwise refinement.**

e.g. a structure diagram of an alarm app:



A detailed structure diagram showing breakdown of tasks



ALGORITHM:

An **algorithm** is a sequence of steps for solving a problem. In general, an 'algorithm' is the name given to a defined set of steps used to complete a task. For instance you could define an algorithm to make a cup of tea. You start by filling the kettle, and then place a tea bag in the cup and so on. In computer terms, an algorithm describes the set of steps needed to carry out a software task. This mini-web takes you through the topic of algorithm.

Creating a plan

The main stages of planning any solution are:
• understanding the problem
• defining the scope of a solution – the extent of the facilities that the solution will provide
• creating the solution
• documenting the solution
• testing the solution.
Creating the solution

This involves:

• top-down design or stepwise refinement
• Algorithms, which can be represented as
– Program flowcharts
– Pseudo code
• Modules of code
• Menus.

Document the solution

Techniques for documenting an algorithm include:
• Structure diagrams
• Program flowcharts (for symbols, see Table 9.1 in course book).
For documenting the hardware required in a solution while showing much less processing detail, we can use system flowcharts, which have a much larger set of symbols.

Testing and interpreting algorithms

Dry running is the process of thinking through the operation of an algorithm, to test it during design, for troubleshooting and to work out its purpose, if not stated.
A trace table is a tabular record of a dry run. It has a column for each variable, usually in the order in which their values are first assigned. Each row is completed with the values of the variables whenever they change, moving to the next row when necessary.

Algorithm vs Pseudocode :

An algorithm is simply a solution to a problem. An algorithm presents the solution to a problem as a well-defined set of steps or instructions. Pseudo-code is a general way of describing an algorithm.
Pseudo-code does not use the syntax of a specific programming language, therefore cannot be executed on a computer. But it closely resembles the structure of a programming language and contains roughly the same level of detail.

Algorithm:

An algorithm gives a solution to a particular problem as a well-defined set of steps. Algorithms can be expressed using natural languages, pseudocode, flowcharts, etc.

Pseudocode:
Pseudocode is one of the methods that could be used to represent an algorithm. It is not written in a specific syntax that is used by a programming language and therefore cannot be executed in a computer.

There are lots of formats used for writing pseudocodes and most of them borrow some of the structures from popular programming languages such as C, Lisp, FORTRAN, etc.

FLOWCHARTS:

A program flowchart is a pictorial representation of an algorithm. Program flowcharts consist of special symbols: Flowcharts provide another way of showing an algorithm visually.

A flowchart is a diagram representing the operations involved in a process. It consists of a combination of:

1. **Symbols** (or **boxes**) to represent the operations. There are standard shapes for the different types of operations which should be remembered and used.

2. **Messages** in the symbols. These state briefly what operations are:

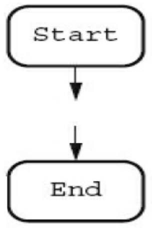
3. **Annotation** This is the explanation given in the margin (Usually at the right of the flowchart). It allows the messages to be kept brief.

4. **Lines** connecting the symbols. These lines may have arrows.

There are two completely different types of flowchart.
1. System flowcharts. These represent the operation on data in a system.
2. Flowcharts of algorithm. These represent the sequence of operations and include program flowcharts.

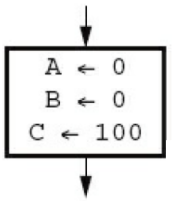
A **FLOWCHART** shows diagrammatically the steps required for a task (sub-system) and the order that they are to be performed. These steps together with the order are called an **ALGORITHM**.
Flowcharts are an effective way to communicate the algorithm that shows how a system or sub-system works.

Standard flowchart symbols:



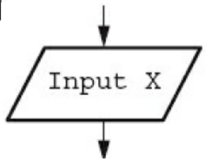
Begin/End:
Terminator symbols are used at the beginning and end of each flowchart.

Process:
Process symbols are used to show when values are assigned to an item/variable like an assignment in pseudocode.



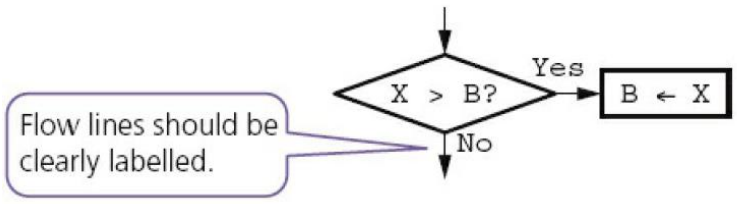
Input/Output:

Input/Output symbols are used show input of data and output of information.



Decision:

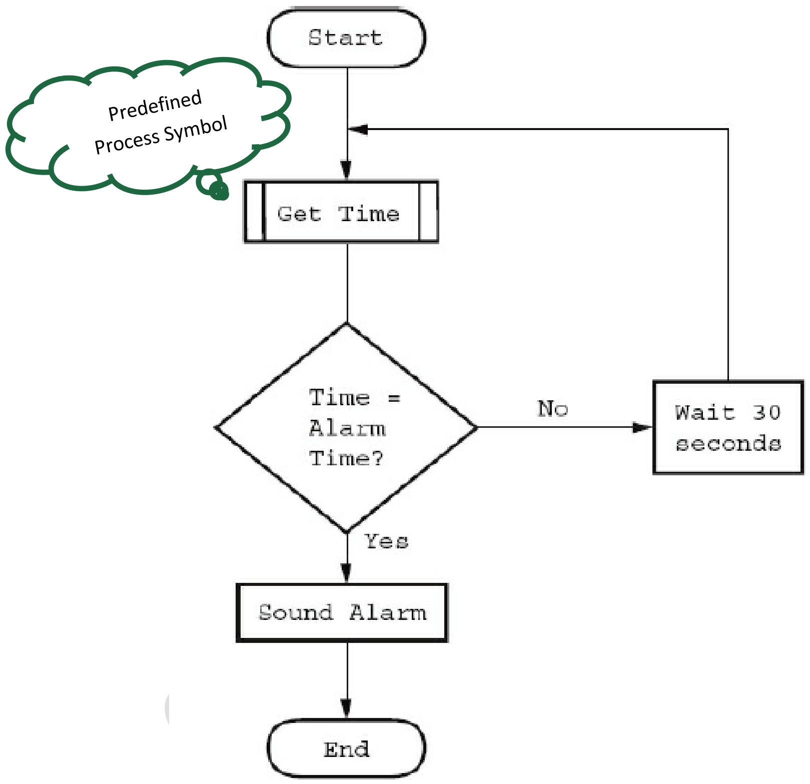
Decision symbols are used to decide which action is to be taken next. These can be used for selection and repetition/iteration.

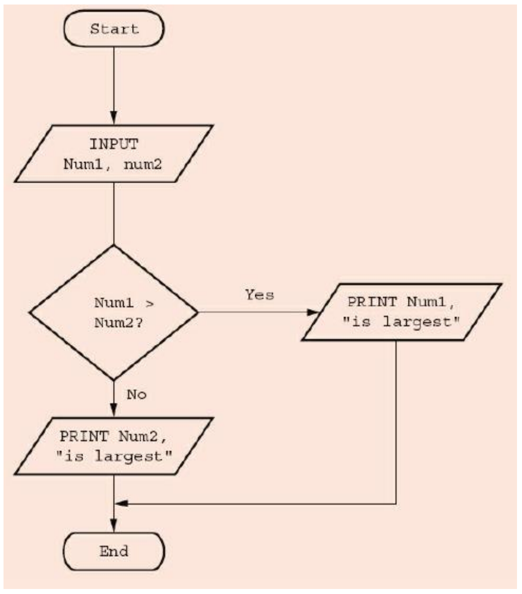


Flow lines

Flow lines are used to show the direction of flow which is usually, but not always, top to bottom and left to right

Example of Flowchart with Loop:





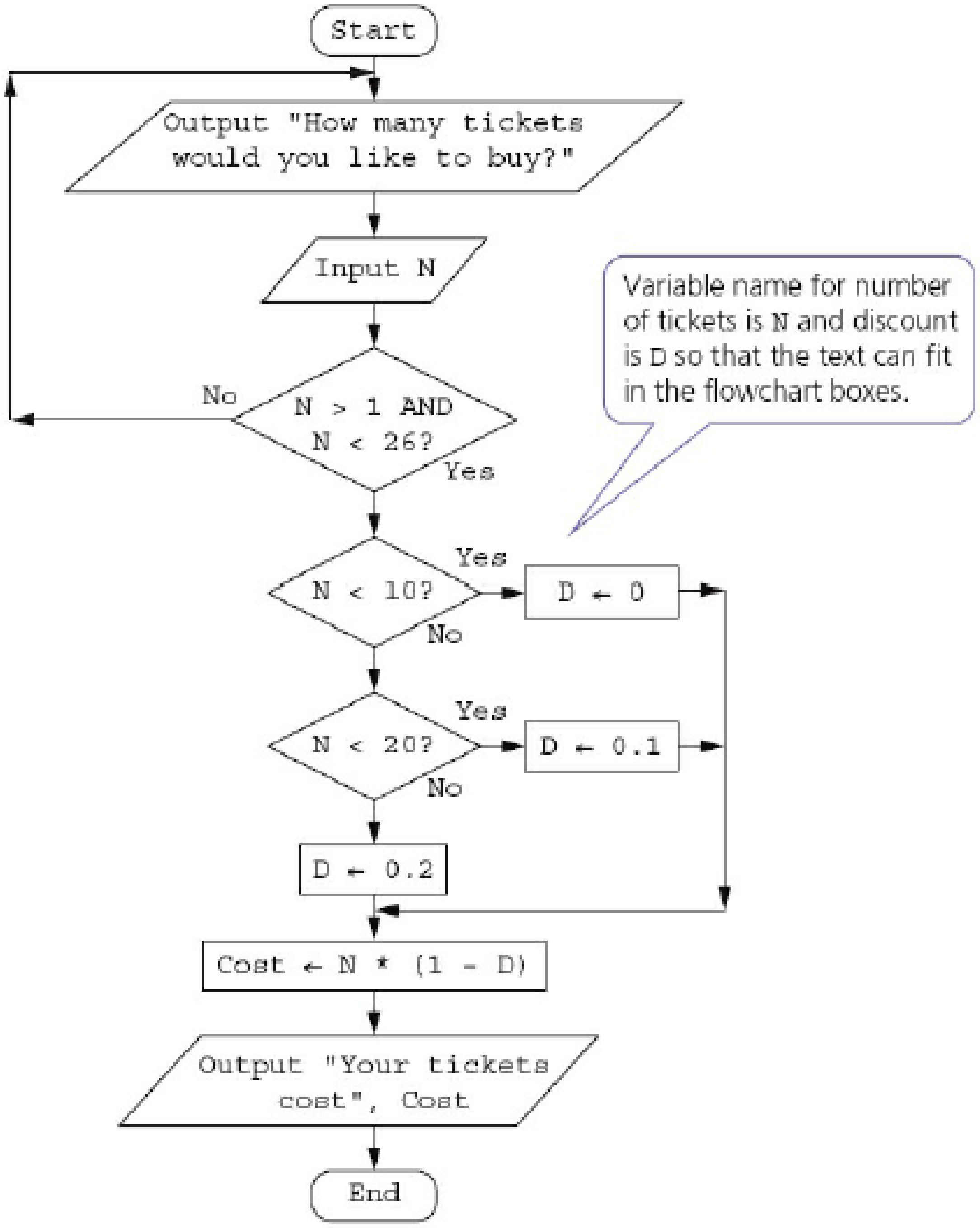
(Pseudocode example)

```
REPEAT
        Wait 10 seconds
Get time
UNTIL time = AlarmTime
Sound Alarm
```

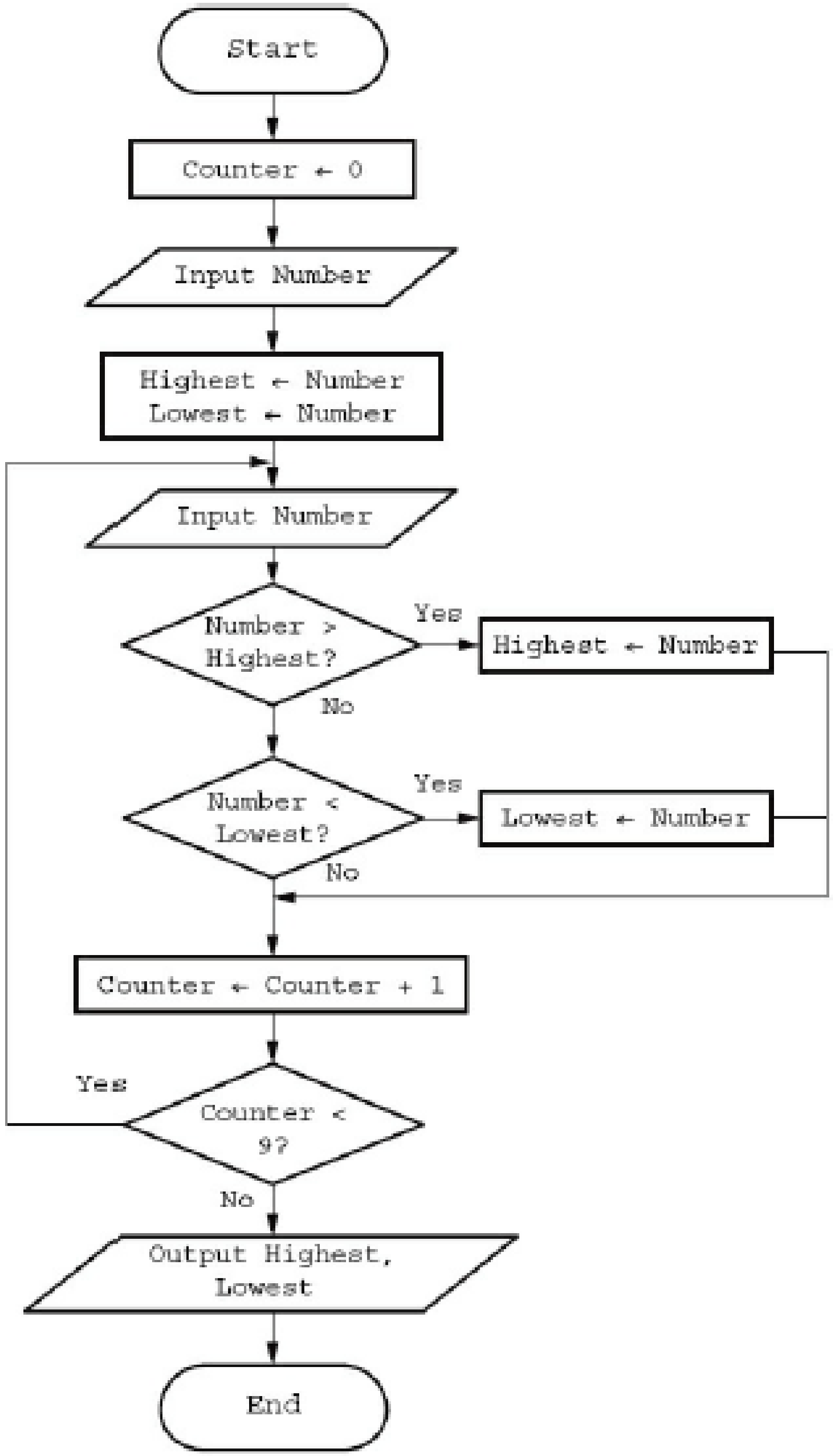
```
INPUT num1, num2
IF num1 > num2
THEN PRINT "num1 is largest"
ELSE PRINT "num2 is largest"
END IF
```

Example 1:

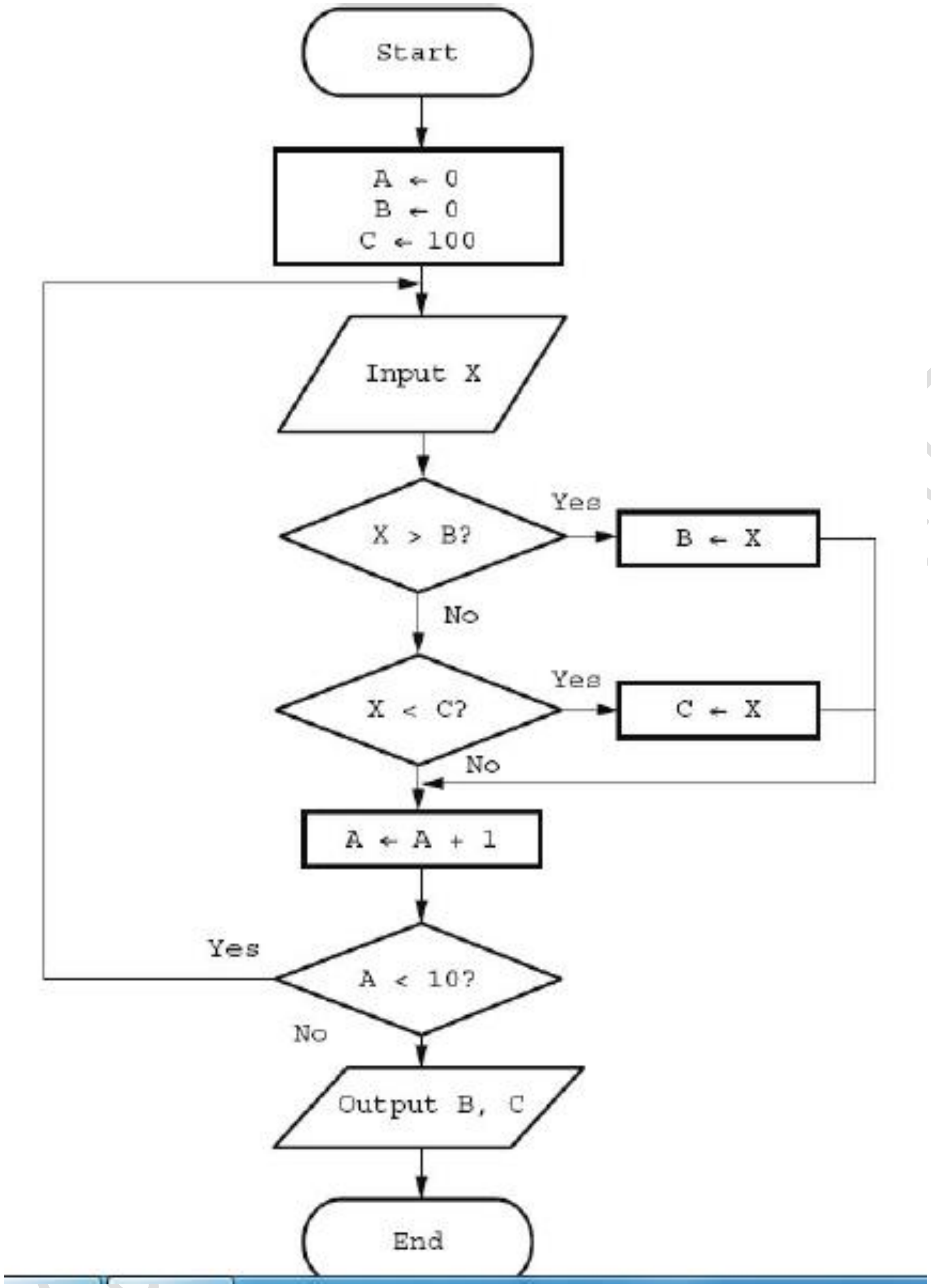
Tickets are sold for a concert at $20 each, if 10 tickets are bought then the discount is 10%, if 20 tickets are bought the discount is 20%. No more than 25 tickets can be bought in a single transaction.


Start
Output "How many tickets would you like to buy?"
Input N
N > 1 AND N < 26?
No
Yes
Variable name for number of tickets is N and discount is D so that the text can fit in the flowchart boxes.
N < 10?
Yes
No
D ← 0
N < 20?
Yes
No
D ← 0.1
D ← 0.2
Cost ← N * (1 - D)
Output "Your tickets cost", Cost
End

Flowchart for the algorithm to calculate the cost of buying a given number of tickets:

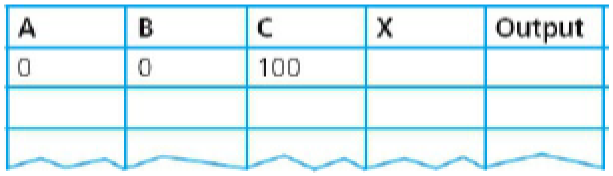


Using Trace Tables:



TRACE TABLE can be used to record the results from each step in an algorithm; it is used to record the value of an item (variable) each time that it changes. This manual exercise is called a **DRY RUN**. A trace table is set up with a column for each variableand a column for any output.

For example :

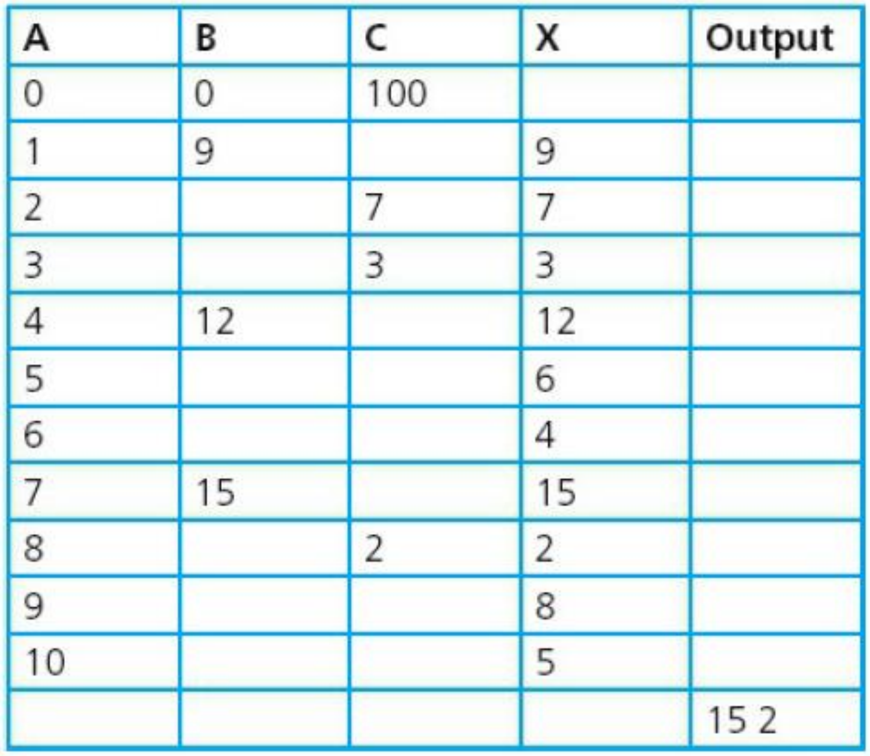


A	B	C	X	Output
0	0	100		

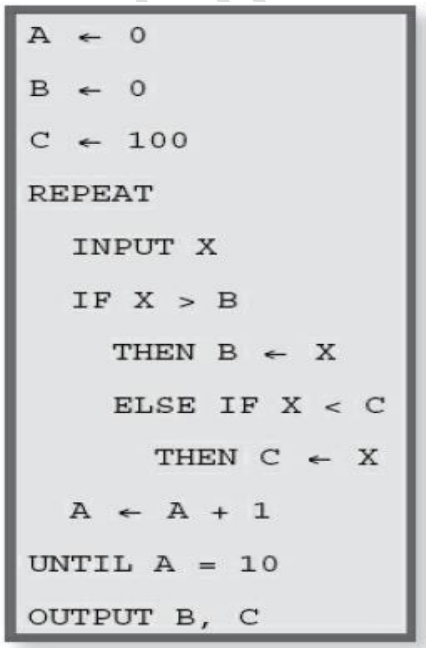
Test data is then used to dry run the flowchart and record the results on the trace table.

Test data: 9, 7, 3, 12, 6, 4, 15, 2, 8, 5

Completed trace table for flowchart



A	B	C	X	Output
0	0	100		
1	9		9	
2		7	7	
3		3	3	
4	12		12	
5			6	
6			4	
7	15		15	
8		2	2	
9			8	
10			5	
				15 2



```
A ← 0
B ← 0
C ← 100
REPEAT
  INPUT X
  IF X > B
    THEN B ← X
    ELSE IF X < C
      THEN C ← X
  A ← A + 1
UNTIL A = 10
OUTPUT B, C
```

It can be seen from the output that the algorithm selects the largest and the smallest numbers from a list of 10 positive numbers.

The same trace table could have been used if the algorithm had been shown using pseudocode.

PSEUDOCODE:
Pseudo code is an outline of a program, written as a series of instruction using simple English sentences.

Pseudo code uses keywords commonly found in high-level languages and mathematical notation. It describes an algorithm's steps like program statements, without being bound by the strict rules of vocabulary and syntax of any particular language, together with ordinary English.

Variable:
Variable is memory location where a value can be stored.

Arithmetic
Use the arithmetic operators.

Operator	Action
+	add
–	subtract
*	multiply
/	divide
^	raise to the power
( )	group

Comparison operators:

Operator	Comparison
>	greater than
<	less than
=	equal
>=	greater than or equal
<=	less than or equal
<>	not equal
( )	group
AND	both
OR	either
NOT	not

Assignment:
Assignment is the process of writing a value into a variable (a named memory location). For example, Count ← 1 can be read as 'Count is assigned the value 1', 'Count is made equal to 1' or 'Count becomes 1'. Another way of indicating assignment is a pseudocode statement such as: set Swapped to False

Initialization:
If an algorithm needs to read the value of a variable *before* it assigns input data or a calculated value to the variable, the algorithm should assign an appropriate initial value to the variable, known as Initialization.

Input:
We indicate input by words such as **INPUT, READ or ENTER,** followed by the name of a variable to which we wish to assign the input value.

Output:
We indicate output by words such as **OUTPUT, WRITE or PRINT,** followed by a comma-separated list of expressions.

Totaling:
To keep a running total, we can use a variable such as Total or Sum to hold the running total and assignment statements such as:
Total ← Total + Number
ADD Number to Total

Counting:
It is sometimes necessary to count how many times something happens.
To count up or increment by 1, we can use statements such as:
Count ← Count + 1
INCREMENT Count by 1

Efficiency:

An algorithm's efficiency can be judged in terms of:
- **Speed**: How quick the algorithm produces the required output.
- **Memory requirements**: How many lines of code the algorithm contains.

Correctness:
Although an algorithm is expected to produce the correct outputs, correctness can still be measured in terms of:

Accuracy:
How many decimal places produce output with greater accuracy (e.g. more decimal places)

Range:
Will the algorithm work with the complete range of inputs? Or can it only deal with positive numbers, whole numbers, numbers below 1 million, etc.

Reliability:
Will the algorithm always produce correct output within the range that it is designed to work? Or are there values which it will not accept (e.g. zero).

Appropriateness: Appropriateness can be measured in terms of:

Length: If the problem is simple then a short algorithm would normally be expected.

Speed: if the output needs to be generated quickly, then the algorithm must be able to generate output quick enough.

Memory requirements: An algorithm controlling a washing machine must not require a lot of memory!

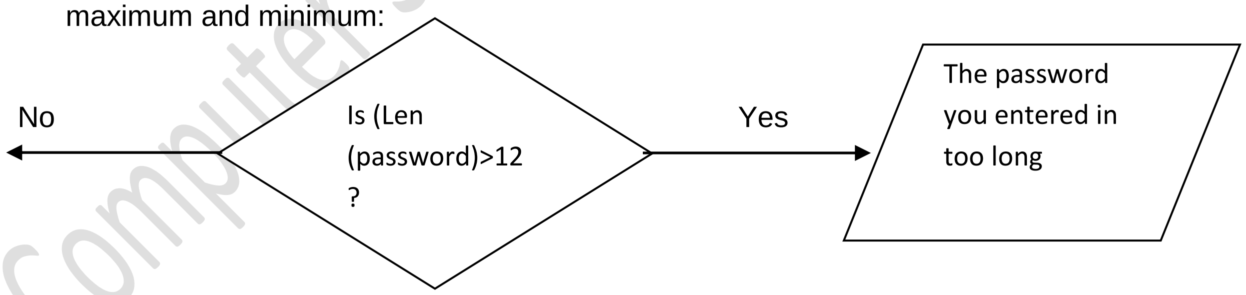
Understand and use assignment statements

What is an Assignment?

An assignment is an instruction in a program that places a value into a specified variable.

Validation:
Validation checks ensure that data entered into the computer is sensible. Data is checked in accordance with a set of rules. The computer's software can validate data while it is being entered into the computer. The main purpose of data validation is to spot an error. This can be done quickly and easily as the process is automated.

- **Range check**. A mathematics exam is out of 100. A simple validation rule that the computer can apply to any data that is input is that the mark must be between 0 and 100 inclusive. Consequently, a mark of 101 would be rejected by this check as being outside the acceptable range.
- **Character check**. A person's name will consist of letters of the alphabet and sometimes a hyphen or apostrophe. This rule can be applied to input of a person's name so that dav2d will immediately be rejected as unacceptable.
- **Format check**. A particular application is set up to accept a national insurance number. Each person has a unique national insurance number, but they all have the same format of characters, 2 letters followed by 6 digits followed by a single letter. If the computer knows this rule then it knows what the format of a NI number is and would reject ABC12345Z because it is in the wrong format, it breaks the rule.
- **Length check**. A NI number has 9 characters, if more or fewer than 9 characters are keyed in then the data cannot be accurate.
- **Existence check**. A bar code is read at a supermarket check-out till. The code is sent to the main computer which will search for that code on the stock file. As the stock file contains details of all items held in stock, if it is not there then the item cannot exist, which it obviously does, therefore the code must have been wrongly read.
- **Length check**
  Code can be added to check that a particular control has a value between an allowed maximum and minimum:



```
IF      LEN (Password) > 12 THEN
        Print ("The password that you have entered is too long")
END
```

References: Wikipedia, PC Mag. Encyclopedia http://www.pcmag.com/encyclopedia/term/52201/subsystem
Computer Science by David Watson & Helen Williams, Zafar Ali Khan www.zakonweb.com