CHAPTER 1: SYSTEMS ANALYSIS AND DESIGN

This is a process of identifying problems of an existing system by analysing it and finding the best solution to such a problem.

System users or the manager would have realized that the Information system is no longer reflective of or satisfying the existing business aims and objectives. This problem could be triggered by many factors, some of which are:
i. System users: they may be dissatisfied with the current system since they are the ones who operate it. They will the sent requests to have a new system or some modification to the existing one.
ii. Top management: they may issue directives in order to meet new organisational objectives. It can also be due to change in management (new manager), new requirements, etc.
iii. The need for improved operating efficiency: Errors in the existing systems may be intolerable, especially to customers.
iv. Changes in technology: new hardware and software may force organisations to change their ways of operation.
v. Change of government policies: new government laws and policies can force organisations to change their systems
vi. The user could have changed his/her mind
Vii. The business might have expanded or due to other reasons

The systems analyst now needs to examine whether the said problem is real by carrying out an in-depth study, after getting permission to conduct a feasibility study.

The systems analyst

This is a person who identifies problems of the existing system and recommends the best solution to such a problem. The duties of a systems analyst are:

- Identifies the problems of the current system.
- Liaises with system users and determine their requirements.
- Finds out facts important to the design of the new system.
- Determines the human and computer procedures that will make up the system.
- Participates in the process of system implementation.

By performing such duties the systems analyst acts as:

i. **A consultant**: can be called or hired to identify problems in a system
ii. **A supporting expert**: draws together professional expertise concerning computer hardware and software and their uses in business.
iii. **An agent of change**: bring new ideas into the organisation

Qualities of a systems analyst

- Must have good oral and written communication skills for all managerial levels of an organisation.
- Must be able to work as a team.
- Must be well educated, with at least a degree.
- Must be well experienced in computers and at top managerial levels.
- Must have good managerial skills.
- Must be a problem solver and see problems as challenges.
- Must be self-motivated.
- Must be well disciplined.
- Must be able to work under pressure and meet deadlines.

SYSTEMS DEVELOPMENT LIFE CYCLE (SDLC)

- It refers to stages that are chronologically completed to develop a new or modified computer system.
- It refers to the stages through which a system develops from 'birth' to 'death', that is, from the moment the system is incepted until it is modified or replaced with a new one.

- Can be shown using the Waterfall Model or the Stepping Stone Model, which encompasses the following stages, in their order;
  - Feasibility Study
  - Data Collection
  - Analysis of the problem
  - System Design
  - System Development and Testing
  - System Implementation
  - System Maintenance

1. Feasibility Study:

-This is a preliminary investigation conducted to determine if there is need for a new system or modification of the existing one.

-The Analyst examines whether a new system is feasible or not.

He assesses the magnitude of this problem and decides the scope of the project.

He examines the problem of the current system and what will be required of the new system.

-It involves evaluation of systems requests from users to determine if it is feasible to construct a new one. Feasibility can be measured by its:

- ✓ **Economic feasibility:** determining whether the benefits of the new system will out-weigh the estimated cost involved in developing, purchasing, installing and maintenance of the new system. The cost benefits analysis is important. Benefits can be tangible and quantifiable, e.g. profits in terms of monetary values, fewer processing errors, increased production, increased response time, etc. Other benefits are intangible, e.g. improved customer goodwill, employee morale, job satisfaction, better service to the community, etc.
- ✓ **Technical feasibility**: determines if the organisation can obtain software, equipment, technology and personnel to develop, install and operate the system effectively.
- ✓ **Schedule feasibility**: a measure of how long the system will take to develop, considering the desired time frame.
- ✓ **Social feasibility**: Will the system be acceptable by the local people, considering their values and norms in their society? This also looks at impacts like loss of jobs,
- ✓ **Legal feasibility**: determines whether the new system will not violet the legal requirements of the state, for instance, laws outlined in the Data Protection Act.
- ✓ **Operational feasibility**: determines whether the current work practices and procedures are adequate to support the system, e.g. effects on social lives of those affected by the system

Thus the analyst must consider the following questions when producing a feasibility study:

- Is the solution technically possible?
- Is the solution economically possible to produce?
- Is the solution economic to run?
- Will the solution be socially acceptable?
- Is skilled workforce available? If not, are training requirements feasible
- How will the system affect employees
- Will profits increase?
- How long will it take to produce the system?
- etc

After carrying out the feasibility study, a feasibility study report must be produced and it contains the following information:

- ✓ A brief description of the business.
- ✓ Advantages and problems of the existing system.
- ✓ Objectives of the new system.
- ✓ Evaluation of the alternative solutions.
- ✓ Development timetable.
- ✓ Management summary
- ✓ Terms of reference. Contents page.
- ✓ Proposed solution, its advantages and disadvantages

2. DATA COLLECTION

The systems analyst collects data about the system. The fact finding methods that can be used include: interviews, record inspection, questionnaire, observations, etc.

i. Interview:

This refers to the face-to-face communication between two or more people in order to obtain information. Interviews can also be done over the phone but the most common ones are face to face. Interviews are done when you want to collect information from a very small population sample.

Advantages of Interviews

- ✓ Effective when gathering information about a system
- ✓ The researcher can ask for clarification on some points that may not be clear.
- ✓ Encourages good rapport between the researcher and the respondent.
- ✓ Non-verbal gestures like facial expressions can help the researcher to determine if the respondent is telling the truth.
- ✓ Information can be collected even from the illiterate since the respondent's language could be used.
- ✓ First-hand information is collected.
- ✓ The researcher can probe to get more information.

Disadvantages of Interviews

- ✓ It is expensive since the researcher has to travel to the interview venue.
- ✓ Difficult to remain anonymous
- ✓ It is time consuming as more time is spent travelling and carrying out the interview.
- ✓ Good interview techniques are required as failure may lead to disappointments.
- ✓ Biased information can be given since the respondent may not tell the truth.

ii. Record inspection:

A fact finding method which involves scrutinising system documents in order to solicit information. Record inspection has the following **Advantages**:

- ✓ Accurate information is collected from system records.
- ✓ Shows how data is collected within the system
- ✓ Shows the exact data that is collected
- ✓ Shows information that must be produced by the system
- ✓ First-hand information is obtained.
- ✓ Gives a good idea of the ways things are actually done rather than how they are supposed to be done.

Disadvantages of record inspection

- ✓ It is time consuming to go through all system records.
- ✓ Manual files can be tiresome to go through and some of the data may be illegible.
- ✓ Confidential files are not given to the researcher
- ✓ Documentation may difficult for an outsider to understand
- ✓ Some records may not be relevant

iii. Questionnaire:

A document with carefully crafted questions to be answered by the respondent by filling on the spaces provided. Questionnaires are used when collecting information from a widely spaced population sample and when collecting information from many people. A questionnaire contains open-ended and closed questions. Open-ended questions are gap filling questions which require the respondent to express his or her own view. Closed questions are guided questions where the respondent just chooses **Yes** or **No**, **True** or **False**, or by just putting a tick on given options. Questionnaires can be distributed personally or by post.

Advantages of questionnaires

- ✓ Effective when collecting a lot of data
- ✓ Questions are very simple to answer.
- ✓ It saves time as questionnaires can be distributes and then collected later.
- ✓ Respondents can fill questionnaires at their own pace.
- ✓ Give guarantees confidential of information, thereby encouraging respondents to give accurate information.
- ✓ They are cheap to use as travel expense can be low

Disadvantages of questionnaires

- ✓ Questions must be well thought out and precise.
- ✓ Some questions are left blank.
- ✓ Some questionnaires may not be returned. Biased information can be collected.
- ✓ Respondents usually do not fill the correct information.
- ✓ It is difficult to analyse information collect using questionnaires.
- ✓ They are expensive to use if the postal system is used.
- ✓ Abusive information can be filled by respondents.

iv. Observations:

It is a fact finding method that involves viewing the actual system in operation by the researcher. The researcher can even take part in operating the system. It is used when the researcher wants to see for himself how the system operates.

Advantages of observations

- ✓ First-hand information is collected.
- ✓ May identify problems that the users did not see
- ✓ Accurate information can be obtained.
- ✓ More reliable information is obtained.
- ✓ Only areas of interest can be observed.
- ✓ The researcher can take part in operating the system thereby getting insight on how the system operates.

Disadvantages of observations

- ✓ People work differently if they feel that they are being observed, therefore inaccurate information can be collected.
- ✓ The researcher may not understand some of the activities being observed.
- ✓ It is time consuming to collect the required information.
- ✓ The researcher may interrupt some of the activities being done.
- ✓ More expensive than other methods

3. The analysis stage:

This is the in-depth study of the system to determine how data flows within the system. It basically involves the following activities:

(a) Information Collection (Using interviews, questionnaires, ..)

(b) Analysis of information collected (using DFDs, flowcharts, . .)
  -produces clear view of the system

(c) Requirements specification
  - List of user requirements
  - Hardware and software requirements

(d) Consideration of alternative solutions
  - Match alternative solutions with requirement specification
  - Justify one solution against others

- flowcharts,
- data flow diagrams
  - concentrates on documenting how data flows in a system,
- structure diagrams
- data dictionary
  - it is a table that stores data about data (metadata), ie. Stores details of data used, including the following
    - ▪ Name of data item (fields or variables)

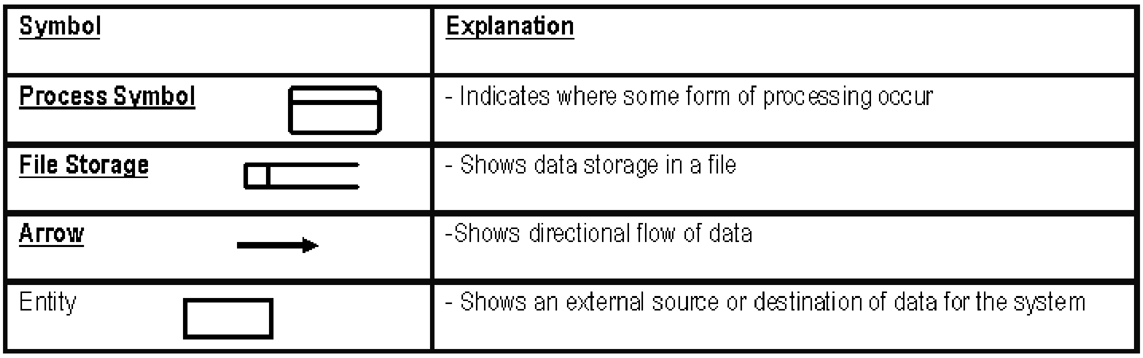
- Data type
- Length
- Validation criteria
- Amount of storage required for each item
- Who owns the data
- Who accesses the data
- Programs which uses the data
- etc

The analysis stage determines whether computerisation will take place or not. The analysis stage also specifies the hardware and software requirements and whether they will use in-house software or outsource the program. The analysis stage looks at the following aspects:

- ✓ Understanding the current system
- ✓ Produce data flow diagrams
- ✓ Identify the user requirements
- ✓ Interpret the user requirements
- ✓ Agree on the objectives with the user
- ✓ Collect data from the current system

Dataflow Diagrams

These are diagrams that show how data moves between external sources, through processes and data stores of a particular system. Dataflow diagrams use the following symbols:

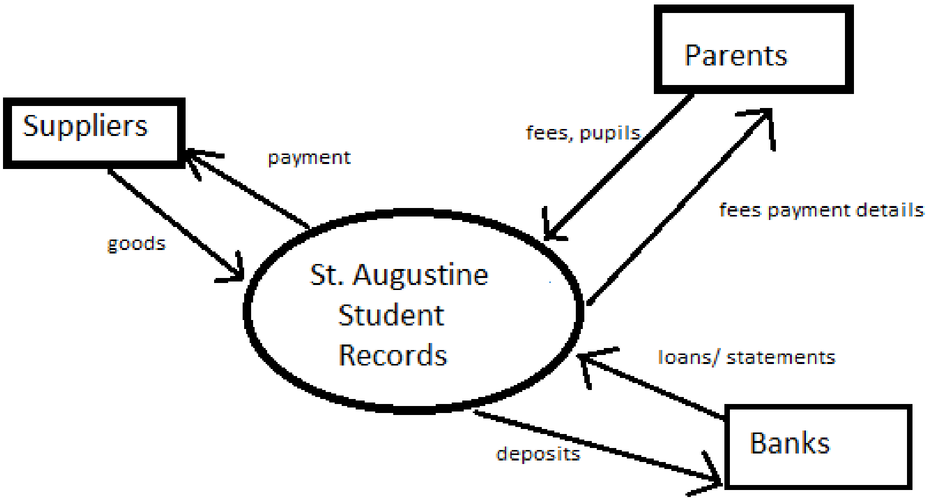


Symbol	Explanation
Process Symbol ▭	- Indicates where some form of processing occur
File Storage ⊏	- Shows data storage in a file
Arrow →	-Shows directional flow of data
Entity □	- Shows an external source or destination of data for the system

(1) Context (zero) Level DFD

The top level DFD which shows one (single) process (which is usually the name of the system) and the entities involved.

It is generalised, with only entities being linked to the system



(2) First Level DFD

The context level DFD is split into detailed processes, entities and data stores and present it as one diagram.

(3) Second Level DFD

The First level DFD is split into more detailed processes, entities and data stores and present it as different diagrams; each first level process has its own separate diagram. Data items between entities and data stores are more detailed (fields are specified in detail, not generalised).

4. Design Stage:

This stage is concerned with the development of the new computer based solution as specified by the analysis stage. The hardware and software requirements are identified and purchased, ready for the construction stage.

Functional diagrams are translated into hierarchical diagrams by the analyst so as to identify what programs are needed and how they relate to one another. The analyst decides on the program structure, program interface and the hierarchy in which programs will be arranged.

The Analyst ensures quality designs, incorporates security measures, designs easy to use input forms, output reports interfaces.

The Database designer fulfills the file requirements. The output is a design specification.

Tools used are DFD, Data Dictionary, Data models, prototypes, system flowcharts: The personnel involved are users, Analysts, Database Designer.

Design stage involves:

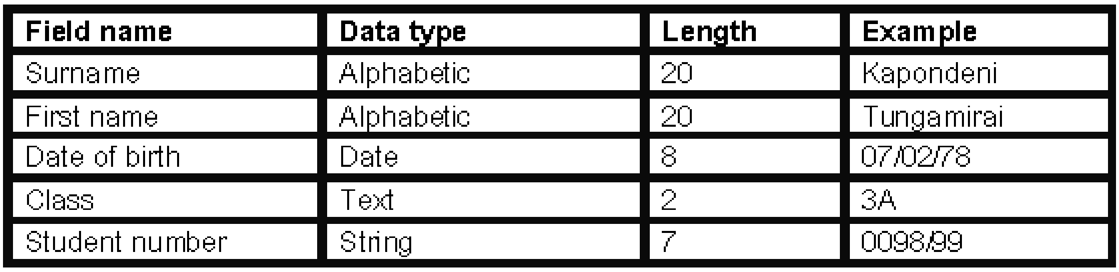
Input Design: includes designing of data entry forms, input methods (e.g. by filling forms, use of barcodes, etc) and the user interface.

Output Design: Includes design of reports, data output formats, screen displays and other printed documents like receipts, etc

File design: Involves designing tables/files, records and validation rules. It also involves determining how data is to be stored and accessed, the media to be used for storage and the mode of file organisation. Fields in each files can be designed, their length and data type, e.g.

File Name: Student File
Storage Location: Hard Disk
Mode of Access: Direct/Random

File Structure:



Field name	Data type	Length	Example
Surname	Alphabetic	20	Kapondeni
First name	Alphabetic	20	Tungamirai
Date of birth	Date	8	07/02/78
Class	Text	2	3A
Student number	String	7	0098/99

It also involves specifying how data is to be sorted and updated.

Program Design: Involves design of systems flowcharts, structure diagrams, dataflow diagrams, algorithms, user interface design, pseudocodes, prototyping, EAR diagramming, etc.

Selection of appropriate hardware: Specify the input hardware, output hardware, processing hardware and speed, storage media, RAM size, etc.

Selection of appropriate software: specifying the type of operating system, programming language or database package and other application packages to be used.

- Design of test plan and strategy,
- Design of the user manual,
- Design of the technical documentation.

In summary, the design stage involves:

- ✓ design the data capture forms/input forms
- ✓ design the screen layouts
- ✓ design output forms and reports
- ✓ produce systems flowcharts and/or pseudocode

- ✓ select/design any validation rules that need to be used
- ✓ design a testing strategy/plan
- ✓ file structures/tables need to be designed/agreed
- ✓ select/design the hardware requirements for the new system
- ✓ select/design the software requirements
- ✓ produce any algorithms or program flowcharts
- ✓ select the most appropriate data verification method(s)
- ✓ interface design(Command-line, GUI, form-based, menu-driven, etc-Pupils to research on these)
    - o When designing interface, the analyst should consider the following;
        - ▪ Who will be using the system?
        - ▪ What knowledge do they have?
        - ▪ How much on-line help is required
        - ▪ What is the information that needs to be shown?
        - ▪ How much information is needed?
        - ▪ What is the best way of showing the information needed
        - ▪ What colours should / should not be used?
        - ▪ What other form of output is sensible
        - ▪ What technology is available
        - ▪ Costs of designing as well as maintainability

PROTOTYPING

Involves building a working but limited model of a new system that will be tested, evaluated and improved if necessary before building the actual system. It involves construction of a simple version of a program which is used as part of the design to demonstrate how the system will work.
It is a mock-up of parts of the system for early evaluation

Reasons for prototyping:
- Gives an idea of the system before development
- enables clear identification of requirements
- allows revision and adjustments before full system is developed

The prototype will have a working interface but may not actually process data
Special software will be used to design input screens and to run the system.
The prototype can then be discarded and the real system designed using other or the same software (**throw away prototype**).
Prototyping can be used at any stage of the SDLC. The prototype can be further refined until the user is satisfied with it and then it is implemented as it is (**Evolutionary prototype**).

Benefits of prototypes:

- ✓ cheaper to setup than alternative methods that could be used to predict what will happen in a system
- ✓ faster to design a system model
- ✓ Gives user the chance to experience the look and feel of the input process and make suggestions where necessary.
- ✓ System is more likely to have fewer or no errors
- ✓ More acceptable to users of the system since they are also involved in the design

Disadvantages of prototyping

- ✓ prototypes can be very expensive to design
- ✓ takes too long to finish system design, especially if the prototype is thrown away

SYSTEM DOCUMENTATION

Documentation refers to the careful and disciplined recording of information on the development, operation and maintenance of a system. Documentation is in two main types: user documentation and technical documentation

(a) User Documentation: It is a manual that guides system users on how to load, operate, navigate and exit a program (system). User documentation contains the following:

- ✓ System/program name.
- ✓ Storage location.
- ✓ System password.

- ✓ Instruction on how to install the program.
- ✓ Instruction on how to operate the system: e.g.
    - o How to quit the program
    - o how to load/run the software
    - o how to save files
    - o how to do print outs
    - o how to sort data
    - o how to do a search
    - o how to add, delete or amend records
    - o print layouts (output)
    - o screen layouts (input)
    - o the purpose of the system/program/software package
    - o error handling/meaning of errors
    - o troubleshooting guide/help lines/FAQs
    - o how to log in/log out

(b) Technical Documentation: This is a manual for programmers which aid them in modifying the system and in correcting program errors. The following are found in technical documentation:

- ✓ Algorithms of the program,
- ✓ Program testing procedures and test data,
- ✓ Sample of expected system results,
- ✓ Program structure showing modules and their relationships,
- ✓ System flowcharts,
- ✓ Programming language used,
- ✓ Program code,
- ✓ program listings,
- ✓ File structures.
- ✓ Validation rules
- ✓ Output formats
- ✓ Bugs in the system
- ✓ Hardware requirements
- ✓ Software requirements

Evaluating Volume Of Data

This involves establishing the amount and type of data involved in a system. This takes the following into consideration:

- Number of forms and reports involved in the systems
- Amount of data and the data types
- How often the data is processed
- Type of processing (online or batch)

NB: **VOLUMETRICS**: Refers to the amount of data to be processed and the characteristics of the users. This may include:

- The number of input documents or online requests
- The number of users and mode of processing (online/batch)

5. Development and Testing

The computer environment is prepared, the programs to be written are done and they are tested, user documentation and training manuals are developed.

Computer environment being prepared: electrical wires, network cables are installed, furniture, air conditioning are in place. The computers are installed and tested.

Programs are written per the program and design specifications. The programs are tested using **walk through** and **group reviews**. The Analyst supervises the writing of training manuals and user documentations. User documentation includes user manuals, user quick reference guides, on-screen help etc.

It also involves the construction and assembling of the technical components that are needed for the new system to operate. This includes preparation of the computer room environment, coding of the computer program using a specific programming language, testing of the coded program, user training (users are trained on how to enter data, search records, edit fields, produce reports, etc).

Testing strategies

First step involves testing of the programs and various modules individually, e.g.

- **Top-Down testing**: program is tested with limited functionality. Most functions are replaced with stubs that contain code. Functions are gradually added to the program until the complete program is tested.
- **Bottom – up testing**:
  Each function is tested individually and then combined to test the complete program.
- **Black-box testing**:
  - Program is regarded as a black box and is tested according to its specification.
  - No account is taken of the way the program is written
  - Different values are entered for variables to determine whether the program can cope with them. This includes standard (typical/normal), extreme (borderline) and abnormal data values.
  - testing will include:
    - Use of extreme, standard and abnormal data
    - Inputting error free data into the system to see if error free outputs can be produced.
    - Inputting data that contains errors into the system to see if the validation procedures will identify the errors.
    - Inputting large quantities of data into the system to test whether or not the system can cope with it.
    - Testing all the regular and occasional processing procedures.
    - Inputting data that contains extreme ranges of information to check that the validation procedures can cope with it.
- **White-box testing**:
  - Each path through the program is tested to ensure that all lines of code work perfectly.
  - Involves testing the program to determine whether all possible paths through the program produce desired results
  - Mostly appropriate if the program has different routes through it, i.e. uses selection control structure and loops
  - Involves testing of logical paths through the code
  - Involves testing of the structure and logic of the program (if it has logical errors)
  - Involves desk checking (dry running)
- **Alpha testing**:
  - The first testing done within the developers company (at owners' laboratory).
  - Testing is done by members of the software company
  - Some errors may still be in existence after alpha testing as the testers are programmers not users.
  - The software version will be unfinished
  - Testers have knowledge of the software and of programming
- **Beta testing**: System testing done after alpha testing; in which the program version is released to a number of privileged customers in exchange of their constructive comments. Mostly similar to the finally released version.

Once a program is tested, it is installed and the analyst can now test it. A very large program must be tested using the following types of tests:

1. **Unit testing**: the process of testing each program unit (sub-routine/module in a suite) singly to determine if it produces expected results.
2. **Integration Testing**: testing to see if modules can combine with each other and work as expected. The whole program is tested to determine if its module integrate perfectly
3. **System testing**: the testing of the whole program after joining the modules to determine if it runs perfectly.
4. **User acceptance testing**: determining if users of the new system are prepared to use it. Usually the final step. It enables identification of some bugs related to usability. User gain the confidence that the program being ushered meets their requirements

Ergonomics: the design and functionality of the computer environment and includes furniture setup, ventilation, security, space, noise, etc. some of the ergonomic concerns include:
Incorrect positioning of the computer facing the window can lead to eyestrain from screen glare. Incorrect sitting positioning can lead to backache. Constant typing with inadequate breaks can lead to RSI. Printer noise can lead to stress. Badly designed software can cause stress. Trailing electricity cables are a safety hazard.

6. System Implementation/ Conversion (Installation/Changeover)

This also involves putting the new computer system into operation, that is, changing from the old system to the new one. It involves file conversion, which is the changing of old data files into the current format. Different changeover methods can be used, and these include:

a. Parallel Run: This involves using of both the old and new system concurrently until the new system proves to be efficient. It involves operating the new and old systems simultaneously until management is confident that the new system will perform satisfactorily. Other workers will be using the old system while others use the old system but doing the same type of job.

- Both the old and new system run concurrently until the new system proves to be efficient.
- Used for very important applications.
- **Costly (expensive)** but the costs are worth paying for.
- can correct the new system while the old system is running
- allows workers to get familiar with the new system
- Output from new system is compared with output from existing system.

Advantages of parallel run

- ✓ Results for both systems are compared for accuracy and consistency.
- ✓ If the new system develops problems, it will be easier to revert to the old one.
- ✓ There is enough time given to determine if the new system produces expected results.
- ✓ Employees have enough time to familiarise with the new system.

Disadvantages of Parallel run

- ✓ Running two systems at the same time is very expensive.
- ✓ Running two systems simultaneously is demanding to employees.
- ✓ It may be difficult to reach a decision when comparing the two systems.
- ✓ There is duplication of tasks which in turn stresses employees

ii. Abrupt (Direct) changeover: Involves stopping using the old system on a certain day and the new system takes over on the next morning. This is suitable for control systems like in chemical plants and for traffic lights.

- new system takes over immediately
- stopping using the old system on a certain day and the new system takes over on the next morning.
- Suitable for control systems like in chemical plants and for traffic lights.
- Very risky because results are so important
- allows no time for training
-No duplication of work, **Cheap** for the organisation, faster to implement, difficult to revert back to the old system if the new one fails.[2]

Advantages of Direct Changeover

- ✓ Reduces cost as of running two systems at the same time.
- ✓ Faster and more efficient to implement.
- ✓ There is minimum duplication of tasks.
- ✓ Enough resources can be allocated to make sure that the new system operates successfully.

Disadvantages of Direct Changeover

- ✓ It will be difficult to revert to the old system if the new system fails.

iii. Phased / Partial conversion: This is whereby the old system is gradually removed while the new system is gradually moved in at the same time. This can be done by computerising only one department in an organisation this month, then the next department in two months’ time, and so on until the whole system is computerised.

Advantages of phased conversion

- ✓ Avoids the risk of system failure.
- ✓ Saves costs since the new system is applied in phases.
- ✓ It could be easier to revert to the old system if the new system fails since only one department will be affected.

Disadvantages of phased conversion

- ✓ It could be very expensive since the organisation will be running two systems but in different departments.

iv. Pilot conversion: one area of organization is converted to the new system while the remainder still uses the old system

- could be even one subject / department
- program is tested in one organisation (or department),
- allows workers to familiarise with the new system on department by department basis.
- effect of one or more problems could be minimised

- **<u>Expensive</u>** to have a pilot study

This is whereby a program is tested in one organisation (or department), and is applied to the whole organisation if it passes the pilot stage. It serves as a model for other departments. A pilot program can then be applied in phases, directly or using the parallel run method.

<u>7. Maintenance/review/evaluation Stage:</u>

This stage is concerned with making upgrades and repairs to an already existing system. Certain sections of the system will be modified with time.
Maintenance can be

(1) **<u>Perfective Maintenance</u>**:

Implies that there room for improving the system even if it is running effectively. For example, improving report generation speed to improve response time. May also incde adding more management information into the system.

(2) **<u>Corrective Maintenance</u>**

Involves correcting some errors that may emanate later, for example, wrong totals, wrong headings on reports, etc. such errors may have been realized when the system has been later a short period of time.

(3) **<u>Adaptive Maintenance</u>**

Involves making the system adapt to changing needs within the organization. For example, changing the system from being a standalone to a multi-user system. May be caused by purchasing of new hardware, changes in software, new government legislation and new tax bands.

<u>NB</u>

Criteria used to evaluate a computer based solution includes the following:

- Were the objectives met? (Successes of the system are compared with set objectives)
- Does it carry out all the required tasks?
- Easiness to use (user friendly)
- Maintainability
- Compatibility with existing systems and hardware
- Offering better performance than the previous one

- Cost effectiveness, etc.

PROJECT EVALUATION AND REVIEW TECHNIQUE (PERT)

-A technique of controlling software project by breaking it into a number of separate activities and stating the order and timeframe for performing each activity.

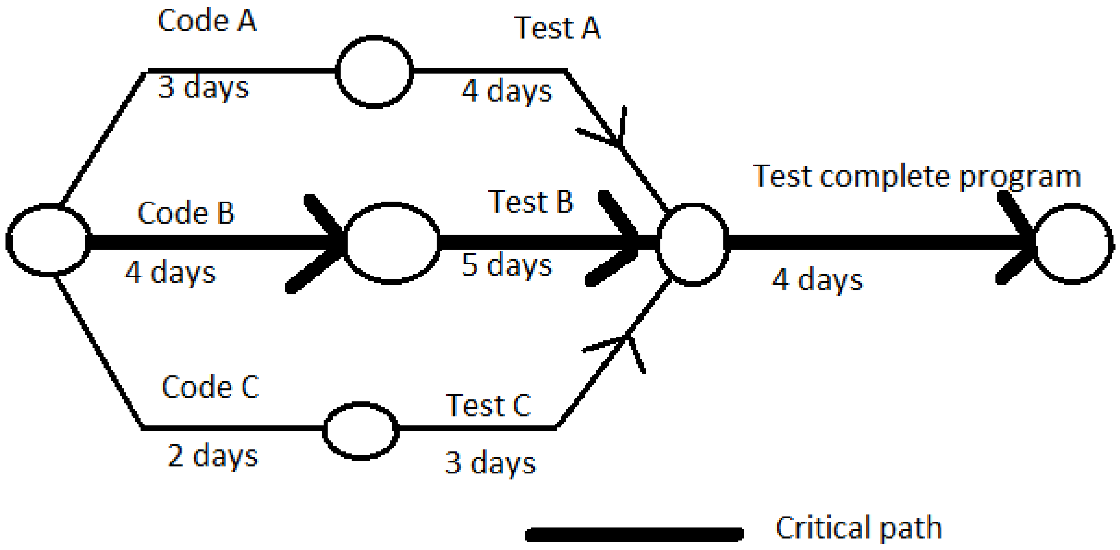
-The order of performing each activity is very important, e.g. coding comes first before testing

-A number of PERT packages can be used

-PERT software packages should show the following information and features:

- Different paths of the project using lines of different value according to length of task
- Arrows showing the order of completion
- Longest journey along the arrows must show the shortest time of completion
- Resources must be allocates at correct times
- Deadlines (bottlenecks) can easily be predicted
- Must have a modelling tool
- Should be able to produce DFDs, ER diagrams
- Should have a library of standard shapes
- Data dictionary must be automatically generated
- Tables can be generated automatically

The activities are all interconnected into a PERT network as given below:



Critical path:

All activities that are critical and are used to determine how long it will take to complete the project; they must not delayed if the project is to finish in time.

Critical Path Analysis:

Analysis of a set of project activities so that delays and conflicts are resolved to reduce project development time to a minimum. It is used to control the sequences of tasks within the project.

Critical Path Method (CPM).

A mathematical algorithm for scheduling and displaying a set of project activities. It includes the following:

- List of all activities required to complete the project
- Time (duration) for each activity will take
- Dependencies between activities (other activities done before others)

CPM calculates the earliest STARTING and latest FINISHING time for each activity.

-It reveals critical activities

-Float time (less critical) – length of time an activity can be delayed or overrun without the whole project being affected , e.g. setting up a printer can be done at the same time as installing the computer but will not take as long.

GANTT CHART

Software for producing Gantt Charts should have some of the following features:

- Should be able to show individual components of tasks
- Should show the earliest starting time
- Should indicate latest ending times
- Should be able to show relationships between components
- Should show the shortest time to finish
- Every aspect should be diagrammatic
- It should be simple to follow
- Should contain review milestones
- Should show percentage of chart finished
- Should automatically generate reports on costs

SYSTEM SECURITY

It is important to keep data secure so that it may not be damaged or get lost. The risks and their solutions are as follows:

Risk	**Solution**
Hardware failure	-Frequent back-up of data, at least one copy to be kept at different locations on daily basis -Log files to be kept for all transactions
Fire	Keep backup file at fireproof safe or storage at an alternative location
Theft	Physical security measures like locking rooms, use security cameras, guards, electric fence, screen gates, etc
Disgruntled employees	Employee checks (ID cards to check workers, careful vetting during employment, instant removal of access right to sacked workers, separation of tasks for workers, educating workers to be aware of security breaches)
Hackers	Usernames & Passwords, firewalls
Viruses	Latest and updated Antiviruses (, firewalls
Floods	Building rooms at higher grounds, waterproof safes for backups

If a hard disc fails, files can be recovered by using the last backup, which is copied on to another hard disc. The log file should be used to update the master file.

During the recovering process, the master file will not be available but the system could be maintained at a lower level of services. Any transaction could be logged and used to update the master file when the system is up and running.

Employee resistance: When a new system is introduced, some employees may resist the change and this could be catastrophic if not handled appropriately. Some fear losing their jobs, of being demoted or being transferred and change of their job description. Resistance can be in the following forms:

- ✓ Through strikes and demonstrations.
- ✓ Giving false information during system investigation.
- ✓ Entering wrong and inappropriate data so that wrong results are produces, etc.

User training:

Once a new system is put in place, existing employees are trained on how to operate the new system, otherwise new employees are recruited. User training can be in the following forms:

i. On the job training: Users are trained at their organisation by hired trainers. This has the following advantages:

- ✓ Learners practice with actual equipment and the environment of the job.
- ✓ Learners can engage in productive practices while on training.
- ✓ This is cheaper for the organisation.
- ✓ Enough practice is gained on how to operate the system.

Disadvantages of on the job training

- ✓ Distractions occur in a noisy office.
- ✓ Instructional methods are often poor.

- ✓ The need to keep up output may lead to hasty teaching

ii. Classroom training: Users are send to other colleges and institutions which offer courses on the subject matter. This could be expensive to organisations since employees take study leave while being paid at the same time. Employees can also be trained on aspects that they will not apply when they finish the course. The gap between what is learnt and what is needed at the job may be too wide.

Considerations when training users:

1. All staff need training that is relevant to their work using the computers. However, others may find it difficult to learn the new system and may resist.
2. Age problem of trainees-older workers takes long to conceptualise concepts
3. Reluctance of employees to learn (use new system)
4. Computer based system means training on the computer
5. Regular updates means new training each time an upgrade is made
6. The type of training should be chosen carefully as it is important
7. Course type with trainer
    - o It may restrict learning times
    - o Can be intimidating to other employees
    - o Difficult to satisfy all trainees' demands
    - o Gives human contact

CONSIDERATIONS FOR CHOOSING A SOFTWARE SOLUTION

1. **Usability**: A system should be user friendly, cut time and have fewer stages in performing a task. Users do not need to consult user manuals all the time the use a system.
2. **Performance:** the system must work in the way intended for and must be reliable (giving few or no errors). It must also be faster in performance, i.e. in processing, displaying records, searching records, etc.
3. **Suitability:** The system should be the real solution to the problem, not that it's the cheapest, the available one, etc. It must also be able to integrate with the existing software in the organisation.
4. **Maintainability:** The system must be easy to upgrade, add new functionality and to make modifications where necessary.

REVIEW QUESTIONS

1. (a) (i) Describe a command-line computer interface [2]

   (ii) State an application for which a command-line interface would be suitable. Justify your choice [2]

   (b) (i) Describe a form-based Computer Interface [2]

   (ii) State an application for which a form-based interface would be suitable. Justify your choice [2]

2. The offices of a government department deals with local taxes in a city. It is decided to develop new software for dealing with the calculation of tax bulls. A systems analyst is employed to develop the software.

   (a) Explain why care must be taken in defining the problem to be solved [2]
   (b) State the methods that the systems analyst can use to find out more information about the problem, giving an advantage of each. [4]
   (c) Explain the importance of evaluating the system against the original specifications. [2]

3. Explain what is meant by:

   (a) Alpha testing, black-box testing, white-box testing, beta testing [8]
   (b) Adaptive maintenance, Corrective maintenance, Perfective maintenance and preventive maintenance [8]

4. The analyst needs to collect information about the current system. State one advantage and one disadvantage of each of the following methods of information collection.

   (a) Questionnaires
   (b) Interviews
   (c) Document Collection
   (d) Observation [8]

5. (a) The analyst has to decide whether to use off-the-shelf or custom – written software. Explain what is meant by:

   (i) Off-the-shelf-software
   (ii) Custom written software [2]

(c) List three advantages and one advantage of using off-the-shelf software rather than custom-written software [4]

6. (a) What is meant by

   i. User documentation
   ii. Technical documentation [2]

   (b) State two items of documentation which would be included in each of the following:

   i. User documentation

   ii. technical documentation [4]

CHAPTER 2: DATABASES

- A database is a single organised collection of structured and related data, stored with minimum duplication of data items so as to provide consistent and controlled data within an organisation.
- Databases can be accessed using different application software.
- Data stored in databases can be accessed by all system users, but with different access rights.
- Databases are designed to meet the information needs of an organisation.
- Database operations may include addition of new records, deletion of unwanted records, amendments to existing records, creation of relationships between files and removal or addition of fields of files.
- Databases ensure controlled redundancy since redundancy cannot be wholly eliminated.

Database terms:

Entity: physical objects like person, patient or events on which information or data is being collected. It can also be an abstract object like a patient record.

Attribute: individual data item within an entity; e.g. date of birth, surname.

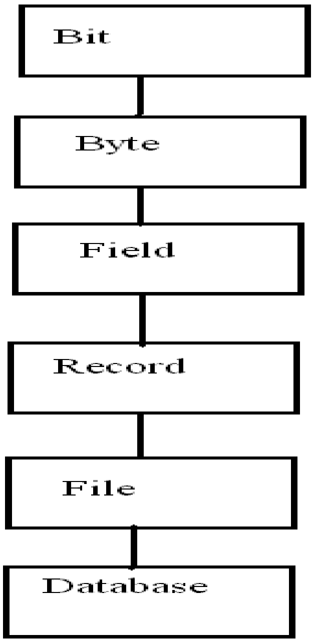
Relationship: links between two different entities or relations (tables). E.g A student stays at St. Augustine's High School. The entities here becomes student and St. Augustine's High School, of which the relationship becomes stays

Data Dictionary:

- It is a table holding information about a database
- A file (table) with description of the structure of data held in a database
- Used by managers when they modify the database.
- Not visible to (used by) general users.
- It maps logical database to physical storage
- Allows existence checks on data to be carried out.
- Stores details of data used, including the following
    - Name of data item (fields or variables)
    - Data type
    - Length
    - Validation criteria
    - Amount of storage required for each item
    - Who owns the data
    - Who accesses the data
    - Programs which uses the data

Flat file: Data stored in a single file (table), allowing simple structuring, e.g. spread sheet database file of student records. Data is stored in rows representing records while columns represent fields. Thus data is stored in a two dimensional format.

Building Block of Computerised Databases



- **Bit**: A single binary digit like 0 or 1.

- **Byte**: A group of eight bits representing a character for example 10010110

- **Field**: A specific category of information in a table (database), for example Surname, Date of Birth, etc

- **Record**: A collection of related fields describing an entity, e.g patient.

- **File**: A collection of related records

- **Database**: A collection of related files

Tuple: it is a row in a relational database, usually denotes a record in a relation.

Types of databases (Database Models)

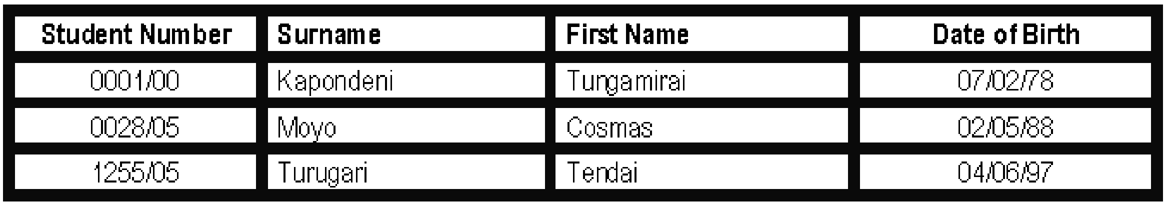
These include relational, hierarchical, network databases and Object oriented databases.

1. Relational databases:

- These are database that organises data in a table format, consisting of related data in another file, allowing users to link the files.
- A table is collection of records stored using rows and column structure.
- Each table is called a **relation**. A relation is a table with columns and rows.
- It only applies to logical structure of the database, not the physical structure.

NB: a RELATION is **NOT** a RELATIONSHIP.

- Each column represents an attribute (characteristic or field). Attribute is a named column of a relation. It corresponds to a characteristic of an entity. They are also called fields.
- Each row represents a record, as shown below.
  - **Degree** is the number of attributes in a relation.
  - **Cardinality** is the number of tuples in a relation
  - Each cell of relation contains exactly one atomic (single) value.
  - Each attribute has a distinct name.
  - Values of an attribute are all from the same domain. Domain is the set of allowable values for one or more attributes.
  - Each tuple is distinct; there are no duplicate tuples.
  - Order of attributes has no significance.
  - Order of tuples has no significance, theoretically



Student Number	Surname	First Name	Date of Birth
0001/00	Kapondeni	Tungamirai	07/02/78
0028/05	Moyo	Cosmas	02/05/88
1255/05	Turugari	Tendai	04/06/97

  - Relational databases organise data in a flexible manner.
  - They are also simple to construct
  - They are easy to use; Easier database design, implementation, management.
  - Ensures structural independence
  - Ensures ad hoc query capability with SQL

Disadvantages

- o Substantial hardware and system software overhead
- o May promote "islands of information" problems
- o However, it may be difficult to come up with relationships.

Database Keys:

- A **simple** key contains a single attribute.
- A **composite key** is a key that contains more than one attribute.
- A **candidate key** is an attribute (or set of attributes) that uniquely identifies a row. A candidate key must possess the following properties:
  - o **Unique identification** - For every row the value of the key must uniquely identify that row.
  - o **Non redundancy** - No attribute in the key can be discarded without destroying the property of unique identification.
- **Super key**: An attribute or a set of attributes that uniquely identifies a tuple within a relation. A **super key** is any set of attributes that uniquely identifies a

row. A super key differs from a candidate key in that it does not require the non-redundancy property.

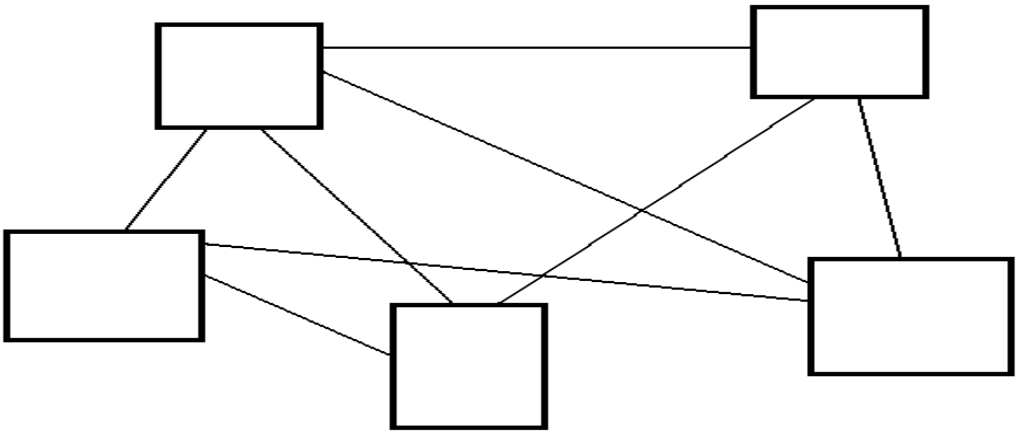
- **Primary key:** It is a candidate key that is used to identify a unique (one) record from a relation. A **primary key** is the **candidate** key which is selected as the principal unique identifier. Every relation must contain a primary key. The primary key is usually the key selected to identify a row when the database is physically implemented. For example, a part number is selected instead of a part description.
- **Foreign key:** A primary key in one file that is used/found in another file. **Foreign key** is set of fields or attributes in one relation that is used to "refer" to a tuple in another relation. Thus it is a filed in one table but also used as a primary key in another table.
- **Secondary Key**: A field used to identify more than one record at a time, e.g. a surname.

*NB: **Attribute**: A characteristic of a record, e.g. its surname, date of birth.

Entity: any object or event about which data can be collected, e.g. a patient, student, football match, etc.

2. Network (Distributed) Databases

It is whereby several computers on the network each hold part of the data which can be shared among the networked computers (users). If data is not available on a user's computer, the user communicates with others on the network to obtain it. Each computer creates its own backup of data resident on it.



- These databases have links that are used to express relationships between different data items.
- They are based on the principle of linked lists
- Data is maintained by a single input.
- There is little duplication of data.
- There is no duplication of inputs.
- Linkages are more flexible.
- Many to many relationships to records are limited

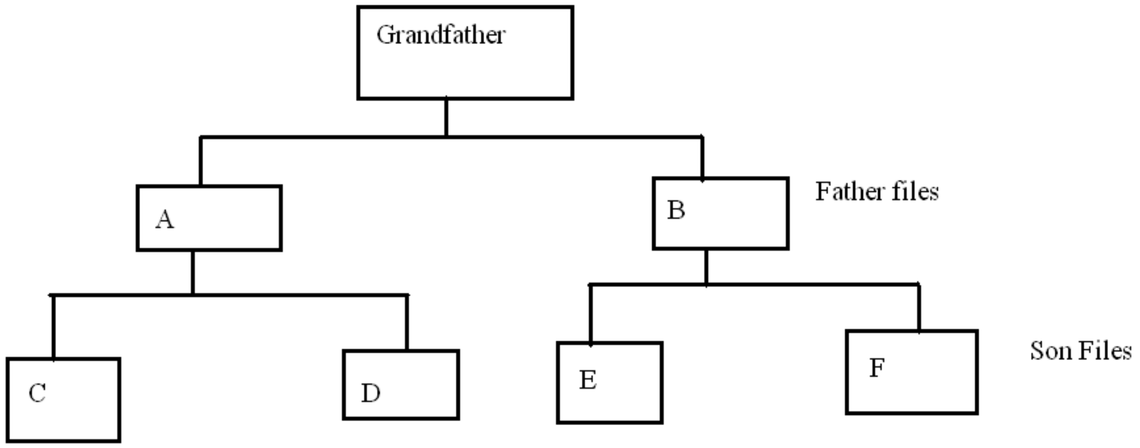
- Handles more relationship types
- Promotes database integrity
- Ensures data independence

However, the system has the following problems:

- Databases may run only on particular computer systems
- System is more complex
- Lack of structural independence

3. Hierarchical database:

Is a database structure in which data is held in a tree structure, indicating different levels of files within the system.
Records are subordinates to other records in a tree structure of database.
Records at the lower level holds more details than their father records
It promotes grandfather, father, and son relationship of records as illustrated below.



- Each father file has one or more son files.
- Each son file has only one father file.
- There are no cross linkages of file records.
- Database security and integrity is ensured
- Complex implementation as it is difficult to access all the files at one time.
- A lot of duplication exists in this type of database structure
- Difficult to manage
- Implementation limitations (no M:N relationship)

4. Object oriented database

An object is a collection of data, an identity, and a set of operations sometimes called methods. An object oriented database stores the data and methods as objects that can be automatically retrieved and shared.An object-oriented database combines database capabilities with an object oriented analysis and design which encompasses features like polymorphism, inheritance and encapsulation. Objects are taken as data. For example a patient database might need to store not only information on name, address and test results but also x rays images.

The object-oriented database (OODB) paradigm is the combination of object-oriented programming language (OOPL) systems and database systems. The power of the OODB comes from the seamless treatment of both persistent data, as found in databases, and transient data, as found in executing programs.

Benefits

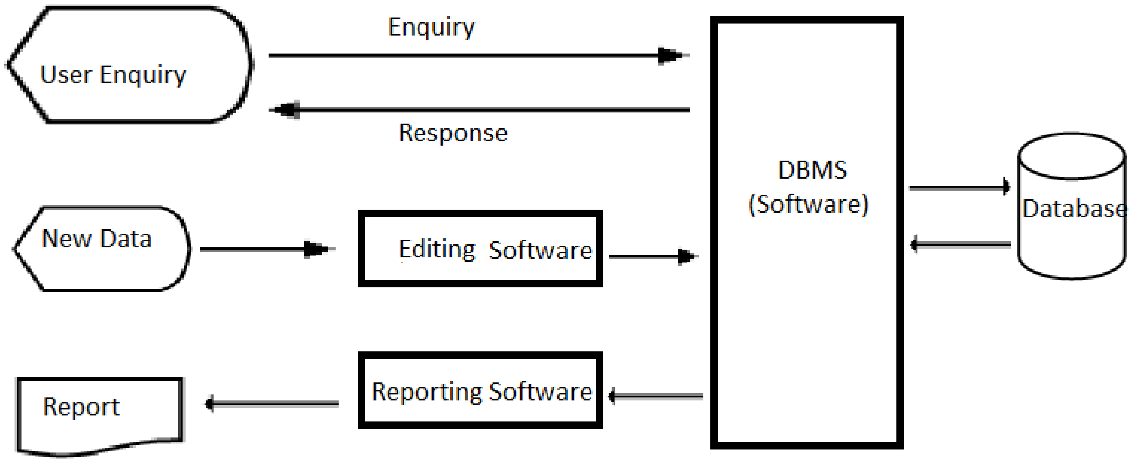
- OODBMS are faster than relational DBMS because data isn't stored in relational rows and columns but as objects. Objects have a many to many relationship and are accessed by the use of pointers, which will be faster.
- OODBMS is that it can be reprogrammed without affecting the entire system.
- Can handle complex data models and therefore is more superior than RDBMS

Disadvantages

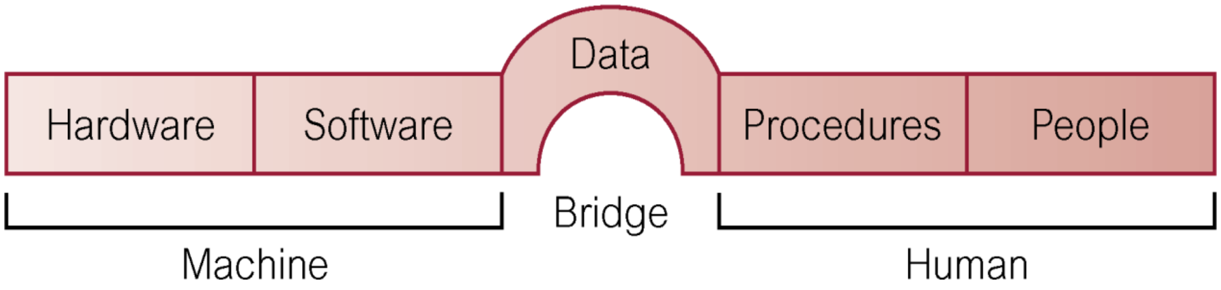
- Pointer-based techniques will tend to be slower and more difficult to formulate than relational.
- Object databases lack a formal mathematical foundation, unlike the relational model, and this in turn leads to weaknesses in their query support.

Database Management System (DBMS)

- It is a complex layer of software used to develop and to maintain the database and provides interface between the database and application programs.
- It allocates storage to data.
- It also allows a number of users to access the database concurrently.
- The DBMS maintains data by:
    - adding new records,
    - deleting unwanted records,
    - amending records.
- Data in databases can be accessed using different programming languages.



Components of DBMS Environment



- **Hardware**
  Can range from a Personal Computer to a network of computers where the database is run.
- **Software**
  DBMS software, operating system, network software (if necessary) and also the application programs.
- **Data**
  Includes data used by the organization and a description of this data called the schema. The data in the database is persistent, integrated, structured, and shared.
- **Procedures**

Instructions and rules that should be applied to the design and use of the database and DBMS. Procedures are the rules that govern the design and the use of database. The procedure may contain information on how to log on to the DBMS, start and stop the DBMS, procedure on how to identify the failed component, how to recover the database, change the structure of the table, and improve the performance.

- **People**:
  Users or people who operate the database, including those who manage and design the database

DBMS Software components

- **Data Dictionary**: this contains names and descriptions of every data element in the database. Descriptions are on how data elements are related to each other. The Data Dictionary (DD) stores data in a consistent memory, reducing data redundancy.
- **Data Languages**: a Special language used to describe the characteristics of a data element placed into the DD. This language is called the DDL
- **Security Software**: provides tools used to shield the database from unauthorised access.
- **Recovery and archiving system**: these allow data to be copied onto backups in case of disaster.
- **Report writers**: these are programs used to design output reports without writing an algorithm in any programming language.
- **Teleprocessing monitors**: Software that manages communication between the database and remote terminals.

DBMS facilities for files:

- Processing of files in serial or sequential order.
- Processing of selected records.
- Retrieval of individual records

DBMS provides security to data in the database by:

- Protecting data against unauthorised access using passwords.
- Safeguarding data against corruption.
- Providing recovery of data due to hardware or software failure

All transaction between the database and the user are done through the DBMS.

Objectives Of DBMS

The main objectives of database management system are data availability, data integrity, data security, and data independence.

Data Availability

Data availability refers to the fact that the data are made available to wide variety of users in a meaningful format at reasonable cost so that the users can easily access the data.

Data Integrity

Data integrity refers to the correctness of the data in the database. In other words, the data available in the database is reliable and consistent data.

Data Security

Data security refers to the fact that only authorized users can access the data. Data security can be enforced by passwords. If two separate users are accessing a particular data at the same time, the DBMS must not allow them to make conflicting changes.

Communicating With the Database

Some databases have their own computer languages. For all the data in databases, data descriptions must be provided. Data Description (Definition) Languages (DDL) are provided as well as the Data Manipulation Language (DML)

*__NB__:

- **DDL**- Refers to data about data (data used to describe data (metadata)).
    - It specifies the data in the database.
    - It defines the structure of the tables.
    - It is used to define the data tables.
    - It specifies the data types of data held.
    - It specifies constraints on the data.
    - contains validation rules for data
- **DML**: Language used by users to (access) retrieve data from databases. It allows user to perform the following
    - Allows storage of data in tables
    - Insert new records
    - Update the database
    - Delete records
    - Modify/edit records
    - Search and retrieve data

A combination of the DDL and the DML is called a Data Sub-Language (DSL) or a Query Language. The most common DSL is the Structured Query Language (SQL)

Each database must have user interface, which may be in the following

Menu-Based Interfaces for Browsing

These interfaces present the user with lists of options, called **menus,** that lead the user through the formulation of a request. It displays a list of options / commands from which the user has to choose one by use of the mouse or keyboard. Menus do away with the need to memorize the specific commands and syntax of a query language; rather, the query is composed step by step by picking options from a menu that is displayed by the system. **Pull-down** and **pop-up** menus are becoming a very popular technique in window-based user interfaces. They are often used in **browsing interfaces,** which allow a user to look through the contents of a database in an exploratory and unstructured manner.

Below is an illustration of a menu driven type of interface:

1. PRINT RECORD

2. DISPLAY RECORD
3. DELETE RECORD
4. EDIT RECORD
5. MY OPTION IS: __

The user has to enter 1, 2, 3 or 4 and then press enter on the keyboard.

Advantages:
It is fast in carrying out task.
The user does not need to remember the commands by heart.
It is very easy to learn and to use.

Disadvantages:
The user is restricted to those few options available and thus is not flexible to use.

Form-Based Interfaces

- A forms-based interface displays a **form** to each user.
- The form has spaces for input/insertion of data
- Insertion fields are provided together with validation checks on data entered.
- It mirrors a hardcopy form.
- Data is entered in strict order.
- The form has explanatory notes /comments on the screen
- It also uses drop-down lists tick boxes, etc
- Each record in the database may have its own form displayed on the screen.
- Users can fill out all of the form entries to insert new data, or they fill out only certain entries, in which case the DBMS will retrieve matching data for the remaining entries.
- Forms are usually designed and programmed for naive (inexperienced) users as interfaces to canned (pre-recorded) transactions.
- Many DBMSs have **forms specification languages,** special languages that help programmers specify such forms.
- Some systems have utilities that define a form by letting the end user interactively construct a sample form on the screen.
- Ensures that no data is missed/left un-entered.
- It is very easy to insert validation checks/routines. (*read Heathcote for more on Form-Based interfaces and other forms of user interfaces)*

Application: Ordering goods online, applying for membership online, applying for an e-mail address online, completing postal order forms, etc. It ensures that only the relevant information is captured/entered.

Command Driven Interface:

- This is an interface which allows the user to type the command to be carried out by the computer through the keyboard as supported by MS-DOS.
- The instruction is typed in at the command prompt
- Commands can be typed individually or can be combined to make a command sequence
- User must know and understand the commands fully.
- The commands are abbreviated and short e.g. Del (for delete), copy, print, e
- The user has to remember the commands to be typed when performing a specific task.
- An example of a program that uses command driven interface is Microsoft Disk Operating System (MS-DOS).
- Application: Technician who maintains a computer system, which requires access to the whole system and its faster to access required information since it is manipulated directly
- Application: Can be used in telnet systems

Advantages

- It saves disk storage space since there are no icons and less graphics involved.
- It is very fast in executing the commands given once the user mastered the commands.
- It saves time if the user knows the commands by heart.

Disadvantages

- It takes too long for the user to master all the commands by heart.
- It is less user friendly.
- More suited to experienced users like programmers.
- Commands for different software packages are rarely the same and this will lead to mix-up of commands by the user.

Graphical User Interfaces

A graphical interface (GUI) typically displays a schema to the user in diagrammatic form, which can implemented using Windows, Icons, Menus and Pointers (WIMP). It is suitable for inexperienced users. The user can then specify a query by manipulating the diagram. In many cases, GUIs utilize both menus and forms. Most GUIs use a **pointing device,** such as a mouse, to pick certain parts of the displayed record.

Advantages of GUI

- It is faster to give commands by just clicking.
- It is easier for a novice (beginner) to use the system right away. It is user friendly (this is an interface that is easy to learn, understand and to use).

- There is no need for users to remember commands of the language.
- It avoids typing errors since no typing is involved.
- It is easier and faster for user to switch between programs and files.
- A novice can use the system right away.

Disadvantages of GUI

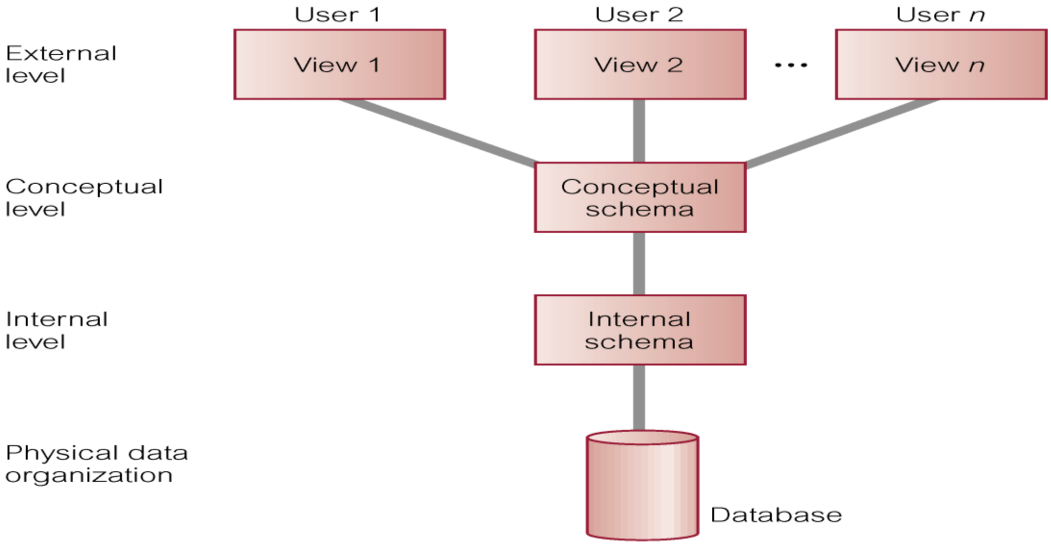
- The icons occupy a lot of disk storage space that might be used for storage of data.
- Occupy more main memory than command driven interfaces.
- Run slowly in complex graphics and when many windows are open.
- Irritate to use for simple tasks due to a greater number of operations needed

DBMS structure (views/schema)

View: Refers to how a user sees data stored in a database. Each user has his/her own view of data, e.g. a standard database user can be restricted from seeing (viewing) sensitive data that only managers can view. These two have thus different views of the database.

- A relation that does not necessarily actually exist in the database but is produced upon request, at time of request.
- Contents of a view are defined as a query on one or more base relations.
- Views are dynamic, meaning that changes made to base relations that affect view attributes are immediately reflected in the view.
- Provides powerful and flexible security mechanism by hiding parts of database from certain users.
- Permits users to access data in a customized way, so that same data can be seen by different users in different ways, at same time.
- Can simplify complex operations on base relations.
- A user's view is immune to changes made in other views.
- Users should not need to know physical database storage details.

Schema: Refers to the overall design of the database. It can be a collection of named objects. Schemas provide a logical classification of objects in the database. A schema can contain tables, views, functions, packages, and other objects.
Sub-schema: describe different views of database.
It consists of three levels/abstractions: External, conceptual and internal levels

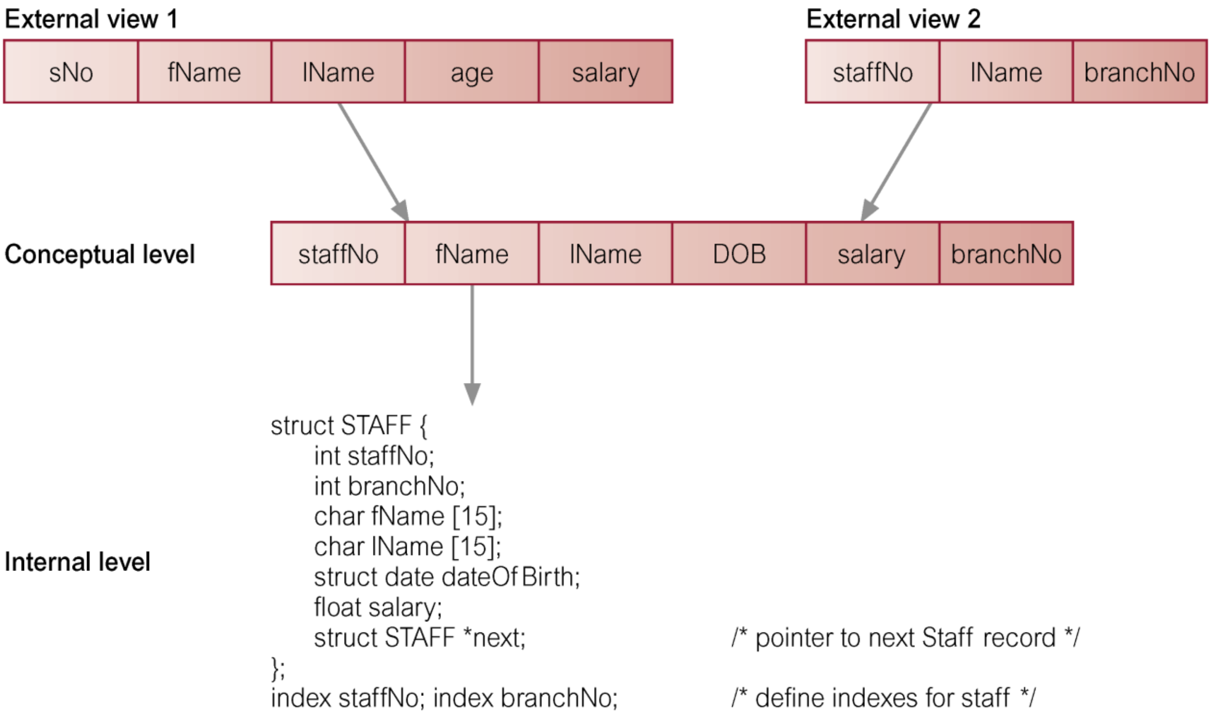


1. **External level (view)**
   - Users' view of the database.
   - It refers to different views of the data seen by each user of the database.
   - Each user of a database has his/her own views of the database, e.g. general database users see and describe the database differently as done by the managers because they have different views of the same database.
   - Describes that part of database that is relevant to a particular user.
   - Thus other users cannot view some data that the managers can views.

2. **Conceptual (logical) Level (view)**
   - It is seen as an integration of all user views of the data
   - This is an abstract representation of the whole database.
   - Describes what data is stored in database and relationships among the data
   - Describes the data as seen by the application that is making use of the DBMS
   - Involves identification of entity types, unique identifiers,
   - Logical level describes what data are stored in the database
   - Describes what relationships exist among those data.
   - Logical level describes the entire database in terms of a small number of simple structures.
   - Database administrator uses the logical level of abstraction.

3. **Internal level (view)**
   - Physical representation of the database on the computer.
   - Describes how the data is stored in the database
   - It refers to the structure used for storage of data.
   - Also called the physical level
   - It encompasses the logical arrangements of the data for storage.
   - Describes how data will be stored on the physical media
   - The physical level describes complex low-level data structures in detail

The diagram below illustrates different view levels of a database;



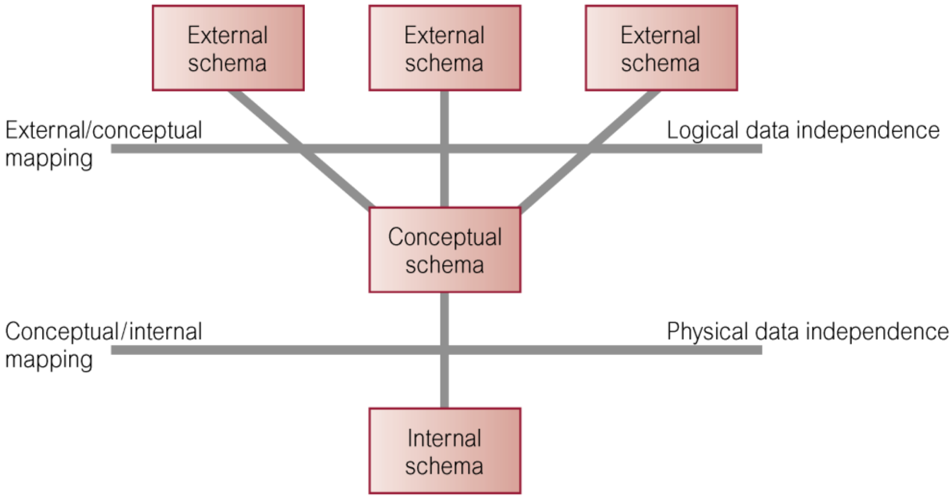
```
struct STAFF {
    int staffNo;
    int branchNo;
    char fName [15];
    char lName [15];
    struct date dateOfBirth;
    float salary;
    struct STAFF *next;              /* pointer to next Staff record */
};
index staffNo; index branchNo;       /* define indexes for staff */
```

Data Independence

Data independence means that programs are isolated from changes in the way the data are structured and stored. Data independence is the immunity of application programs to changes in storage structures and access techniques. For example if we add a new attribute, change index structure then in traditional file processing system, the applications are affected. But in a DBMS environment these changes are reflected in the catalogue, as a result the applications are not affected. Data independence renders application programs immune to changes in the logical and physical organization of data in the system.

Logical organization refers to changes in the Schema. Example adding a column or tuples does not stop queries from working.

Physical organization refers to changes in indices, file organizations, etc


External schema
External schema
External schema
External/conceptual mapping
Logical data independence
Conceptual schema
Conceptual/internal mapping
Physical data independence
Internal schema

- **Physical data independence**
  - Means the applications need not worry about how the data are physically structured and stored.
  - Refers to immunity of conceptual schema to changes in the internal schema.
  - Internal schema changes (e.g. using different file organizations, storage structures/devices) should not require change to conceptual or external schemas.
  - Physical data independence is the ability to modify physical schema without causing the conceptual schema or application programs to be rewritten.
- **Logical data independence**
  - It is the ability to modify the conceptual schema without having to change the external schemas or application programs.
  - Refers to immunity of external schemas to changes in conceptual schema.
  - Conceptual schema changes (e.g. addition/removal of entities), should not require changes to external schema or rewrites of application programs

<u>Passive Database Management System</u>

- It is program-driven, i.e., users query the current state of database and retrieve the information currently available in the database.
- Traditional DBMS are passive in the sense that they are only invoked by user or application program operations.
- Applications send requests for operations to be performed by the DBMS and wait for the DBMS to confirm and return any possible answers.
- The operations can be definitions and updates of the schema, as well as queries and updates of the data.
- The scope of a query in a passive DBMS is limited to the past and present data.

<u>Active Database Management System</u>

- These are data-driven or event-driven systems. The users specify to the DBMS the information they need.

- If the information of interest is currently available, the DBMS actively monitors the arrival of the desired information and provides it to the relevant users.
- The scope of a query in an active DBMS is not only limited to the past and present data, but also includes future data.
- An active DBMS reverses the control flow between applications and the DBMS instead of only applications calling the DBMS, the DBMS may also call applications in an active DBMS.
- Active databases contain a set of active rules that consider events that represent database state changes, look for TRUE or FALSE conditions as the result of a database predicate or query, and take an action via a data manipulation program embedded in the system.

The Database Administrator (DBA)

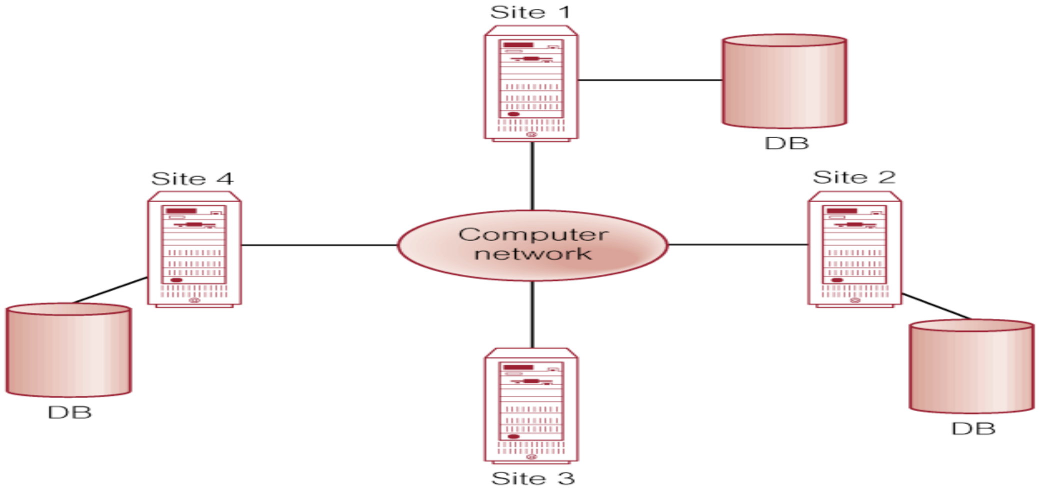
This is a person appointed to manage the database and ensures that the database meets the needs of the organisation. The DBA is supposed to have software and managerial skills.

The responsibilities of the DBA are:

- Ensuring that the database meets the needs of the organisation.
- Setting up the database together with the programmers.
- Control, manage and maintain the database.
- Define, implement and control database storage.
- Ensure that policies and procedures are established.
- Guarantee effective production, control and use of data.
- Define the strategy of backup storage and recovering from system breakdown.
- Supervise amendments to the database.
- Ensures that the data is secure from unauthorised access.
- To control the database environment
- To standardize the use of database and associated software
- To support the development and maintenance of database application projects
- To ensure all documentation related to standards and implementation is up-to-date

Distributed Database

A logically interrelated collection of shared data (and a description of this data), physically distributed over a computer network.



Distributed DBMS (DDBMS)

It is a Software system that permits the management of the distributed database and makes the distribution transparent to users.

Some initial motivations:

- The development of computer networks promotes decentralization.
- In a company, the database organization might reflect the organizational structure, which is distributed into units. Each unit maintains its own database.
- Sharing of data can be achieved by developing a distributed database system which:
    - makes data accessible by all units
    - stores data close to where it is most frequently used.

DDBMS - characteristics

- Collection of logically-related shared data.
- Data split into fragments.
- Fragments may be replicated.
- Fragments/replicas allocated to sites.
- Sites linked by a communications network.
- Data at each site is under control of a DBMS.
- DBMSs handle local applications autonomously.
- Each DBMS participates in at least one global application.

Advantages of DDBMSs

- Reflects Organizational Structure
- Improved Sharing and Local Autonomy
- Improved Availability: A failure does not make the entire system inoperable
- Improved Reliability: Data may be replicated
- Improved Performance: Data are local to the site of "greatest demand"
- Modular Growth: easy to add new module

Disadvantages of DDBMSs

- More complex
- Cost: Especially in system management

- Security: network must be made secure
- Integrity Control More Difficult
- Database Design More Complex: due to fragmentation, allocation of fragments to a specific site.

(read Heathcote for more on record locking, Open Systems and ODBC, Client – Server databases, etc)

Advantages of Databases

- **Reduces data duplication**: Avoids repetition of same records being stored more than once in the database. This is because records are linked to each other allowing data stored in all tables to be used through accessing one table
  - Duplication of data means same data being stored more than once.
  - This can also be termed as data redundancy. Data redundancy is a problem in file-based approach due to the decentralized approach. The main drawbacks of duplication of data are:
    - Duplication of data leads to wastage of storage space. If the storage space is wasted it will have a direct impact on cost. The cost will increase.
    - Duplication of data can lead to loss of data integrity; the data are no longer consistent. Assume that the employee detail is stored both in the department and in the main office. Now the employee changes his contact address. The changed address is stored in the department alone and not in the main office. If some important information has to be sent to his contact address from the main office then that information will be lost.
- Validation checks are made on data during entry thereby reducing data entry errors.
- Searching and retrieval of data is very fast.
- **Ensures data independence**: A change in the program structure or view does not affect data stored in tables. Data independence means independence between application program and the data. The advantage is that when the data representation changes, it is not necessary to change the application program

NB: Data Dependence: This means the application program depends on the data. If some_modifications have to be made in the data, then the application program has to be rewritten.

- **Improves security of data**: Access to some data can be controlled because each user has own view of data. The DBMS can use **access rights (levels) for each user** when accessing data, preventing users from seeing data not of their level. **Regular backups** can be made to the data files automatically by the DBMS to alternative devices. Usernames and passwords can be used to protect data from unauthorised access, record **locking** during updating process, **encryption** of database, etc
- Less likelihood of data getting lost.
- Record structure can be easily modified if the need arises.

- Files can be linked together making file updating easier and faster. Reduces data redundancy. Redundancy means duplication of data. Data redundancy will occupy more space hence it is not desirable as it will be more expensive to the organisation.
- Data can be secured from unauthorised access by use of passwords.
- Users can share data if the database is networked. Duplication of records is eliminated.
- Ad hoc reports can be created easily.
- **Improves Data Integrity**: refers to the correctness of data stored in databases. Data accessed will be similar to all users, removing contradictions caused by duplicates of records with different data values. This is because most of the information is stored only once. Integrity is also enhanced as data is protect from wrong/inappropriate processing thereby leading to users trusting the correctness of data
- Sorting of records in any order is very fast
- Removes data inconsistency: inconsistency means different copies of the same record will have data with different values.

Disadvantages of databases

- If the computer breaks down, you may not be able to access the data.
- It is costly to initially setup the database.
- Computer data can be easily copied illegally and therefore should be password protected.
- Takes time and costs to train users of the systems.
- Expensive to employ a database administrator who will manage the database

Individuals are concerned (worried) with their data held in Computers. This is because of:

- Some people do not want others to see their details (personal data)
- Individuals may be targeted because of their property or wealth
- May lead to comparison with other people's details, which may negate relationships with friends and colleagues
- May lead to blackmail if the data stored is wrong
- Some of the data may be wrong
- Some of the data may be used for other purposes against the owner
- May lead to identity theft

Relational Database Vs Flat File

Relational database	**Flat file**
Less duplication of data as data does not need to in every table	Too much duplication of data since every table repeats data
Offer greater data integrity as there is little chance of getting duplicate of data	No data integrity is guaranteed due to too much duplication of data

Data is available to all users of the system as there are no problems of software incompatibility. Thus users share files	Data not available everywhere as there is no sharing of files
Creates different user views within the DBMS	No different user views are created
Easy to access data from different table due to relationships between the tables	Difficult to access data from different files as the files are not linked
Retrieval of records from different files is faster	Retrieval of records from different files is slower
Better security of records is enhanced	Less security of data from unauthorised access
Promote program data independence	There is program data dependence
There is centralised management of data which is more efficient	No central data management, difficult to manage and less security

Incompatible File Formats

As file-based system lacks program data independence, the structure of the file depends on the application programming language. For example, the structure of the file generated by FORTRAN program may be different from the structure of a file generated by "C" program. The incompatibility of such files makes them difficult to process jointly.

Advantages of computer based systems as compared to manual filing systems.

- Work can be done anywhere, even at offices in different countries.
- It is quicker to transfer files to other offices.
- Fewer staff is employed thereby saving expenses on wages.
- There are fewer chances of files getting lost.
- Less paperwork and storage requirements are needed.
- It is quicker to search needed records.
- It is quicker to sort records in any order using any field.
- It is quicker to cross reference files.
- It is quicker and easier to insert sections of files into reports

However, the introduction of the computer systems means that staff would need new skills, can lead to unemployment, people are likely to work from home, could lead to de-skilling and some health problems will suffice. Can you identify some of the health problems and how they can be solved or minimised?

Structured Query Language

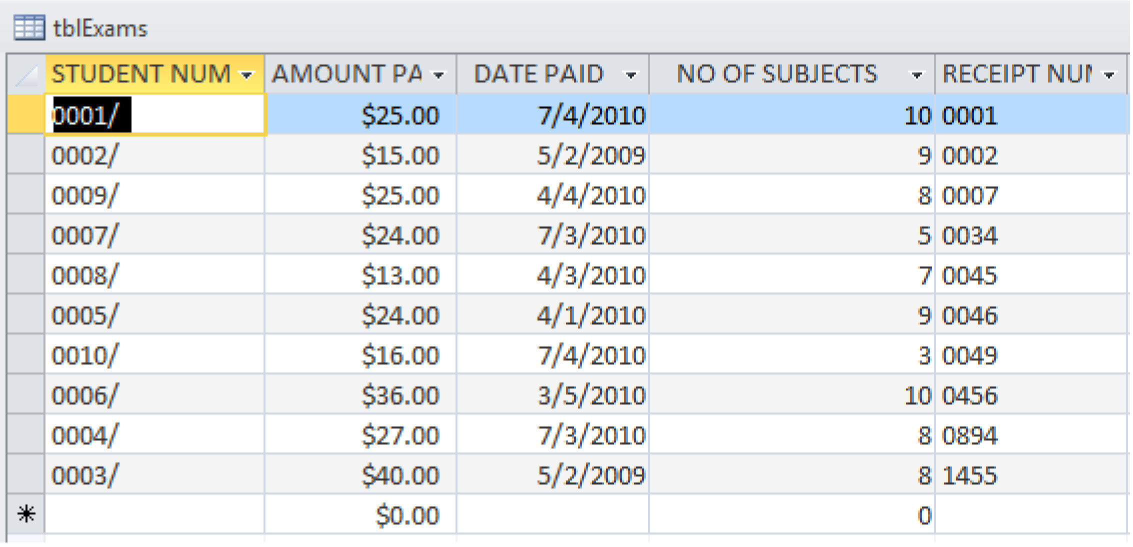
SQL is the standard computer language used to communicate with relational database management systems. A SQL is a standard (de facto) query function for searching data from databases. It is a medium for communication with databases.

SQL commands consist of English-like statements which are used to query, insert, update, and delete data. English-like statements mean that SQL commands resemble English language sentences in their construction and use and therefore are easy to learn and understand. SQL is referred to as nonprocedural database language. Here nonprocedural means that, when we want to retrieve data from the database it is enough to tell SQL what data to be retrieved, rather than how to retrieve it.

A query is a user–request to retrieve data or information using a certain condition. A query is a question used to search and retrieve information is a database. It is a command written in data query language which allows users to access and manipulate data stored in databases. It also allows users to define/ instruct the computer to list or print out selected information from the database, making use of expressions. A query is structured so that the answer is true or false for each record in the database. The result will be a list of all records in the form of a report that returns true to the search criteria. Thus queries are used to interrogate, select and for searching records from the database.

The user specifies a certain condition. The program will go through all the records in the database file and select those records that satisfy the condition. The result of the query will then be stored in form of a table.

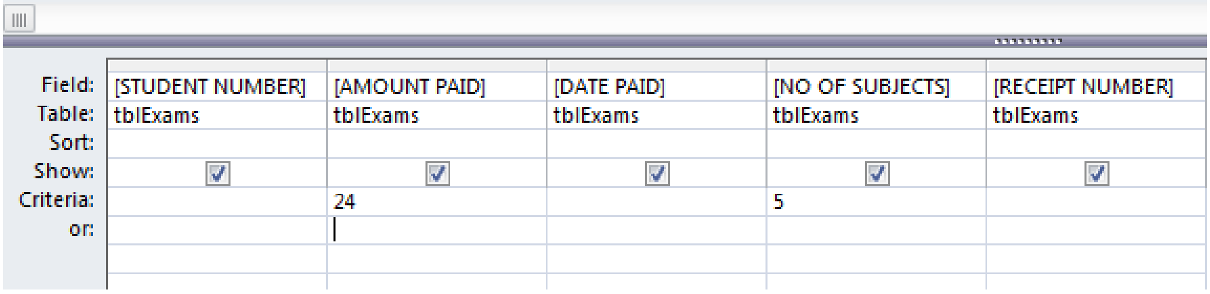
In **Microsoft Access**, users just type the data to be searched like in the table below:



tblExams

STUDENT NUM	AMOUNT PA	DATE PAID	NO OF SUBJECTS	RECEIPT NUI
0001/	$25.00	7/4/2010	10	0001
0002/	$15.00	5/2/2009	9	0002
0009/	$25.00	4/4/2010	8	0007
0007/	$24.00	7/3/2010	5	0034
0008/	$13.00	4/3/2010	7	0045
0005/	$24.00	4/1/2010	9	0046
0010/	$16.00	7/4/2010	3	0049
0006/	$36.00	3/5/2010	10	0456
0004/	$27.00	7/3/2010	8	0894
0003/	$40.00	5/2/2009	8	1455
	$0.00		0	

If one wants to search students who paid $24 and the number of Subjects as 5, he enters the following in the design view of the table query;



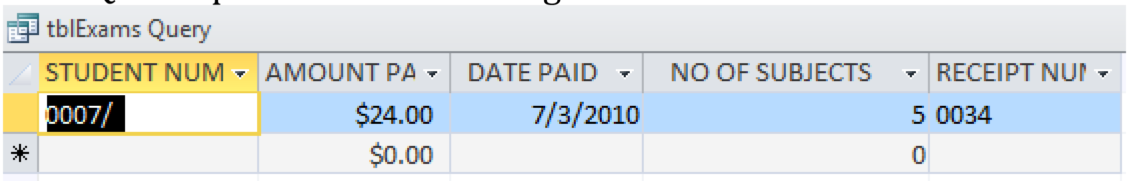
Field:	[STUDENT NUMBER]	[AMOUNT PAID]	[DATE PAID]	[NO OF SUBJECTS]	[RECEIPT NUMBER]
Table:	tblExams	tblExams	tblExams	tblExams	tblExams
Sort:					
Show:	☑	☑	☑	☑	☑
Criteria:		24		5	
or:					

The above can be written in SQL as given below in order to produce the same result:

```
SELECT [ALL / DISTINCT] expr1 [AS col1], expr2 [AS col2] ;
FROM tablename WHERE condition
```

SELECT *tblExams.[STUDENT NUMBER], tblExams.[AMOUNT PAID], tblExams.[DATE PAID], tblExams.[NO OF SUBJECTS], tblExams.[RECEIPT NUMBER]*
FROM *tblExams*
WHERE *(((tblExams.[AMOUNT PAID])=25)* **AND** *((tblExams.[NO OF SUBJECTS])=5));*

The SQL will produce the following result:



tblExams Query

	STUDENT NUM	AMOUNT PA	DATE PAID	NO OF SUBJECTS	RECEIPT NUI
	0007/	$24.00	7/3/2010	5	0034
*		$0.00		0	

SELECT * FROM *tblExams*;
Here the asterisk symbol indicates the selection of all the columns of the table.

SELECT *name, mtest* **FROM** *student* ;
WHERE *class*="1A" **AND** ;
mtest **BETWEEN** *80* **AND** *90*

__NB__: SQL is not only used for searching records from databases. It has commands to delete, insert, print, update, modify, sort data stored in databases.
The three main divisions in SQL are DDL, DML, and DCL. The data definition language (DDL) commands of SQL are used to define a database which includes creation of tables, indexes, and views. The data manipulation commands (DML) are used to load, update, and query the database through the use of the SELECT command. DML commands are usually written in uppercase. Data control language (DCL) is used to establish user access to the database. *(Read **Heathcote** for more SQL, page 308)*

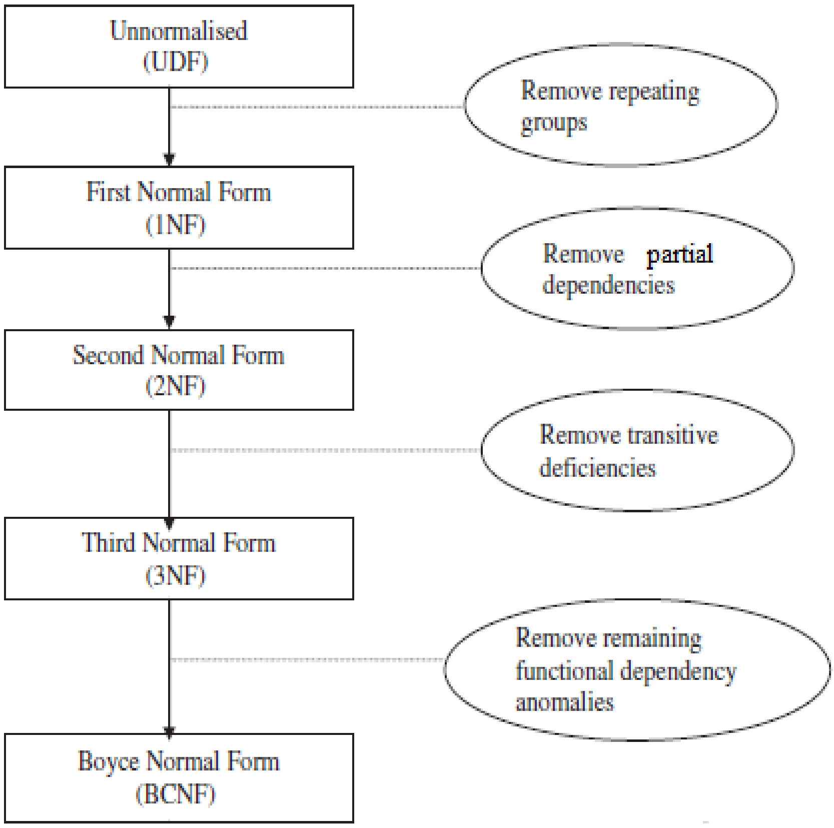
__DATABASE NORMALISATION__

Database normalisation is the process of organising fields and tables of a relational database to minimize redundancy and dependency. Normalization usually involves dividing large tables into smaller (and less redundant) tables and defining relationships between them. The objective is to isolate data so that additions, deletions, and modifications of a field can be made in just one table and then propagated through the rest of the database via the defined relationships.

Normalization is the analysis of functional dependencies between attributes and deciding which attributes should be grouped together in a relation. Data in a table is normalised to a specific form to prevent possible occurrence of update anomalies, basing on functional dependencies among the attributes in the relation. It is done so as to:

- To reduce duplication of data by storing each record/data within the database only once
- Ensure that data stored in databases is consistent.
- Produce controlled redundancies to link tables
- To put data into a more flexible form that is able to accurately accommodate change
- To avoid certain anomalies like updating, insertion and deleting anomalies
- Allow users to make queries relating to data stored in different tables

Normalization stages



1NF - First normal form

- A table is in 1NF if and only if all columns contain only atomic values; that is, there are no repeating groups (columns) within a row. That is, a relation in which the intersection of each tuple and attribute (row and column) contains one and only one value.
- All entries in a field must be of same kind and each field must have a unique name, but the order of the field (column) is irrelevant.
- Each record must be unique and the order of the rows is irrelevant.

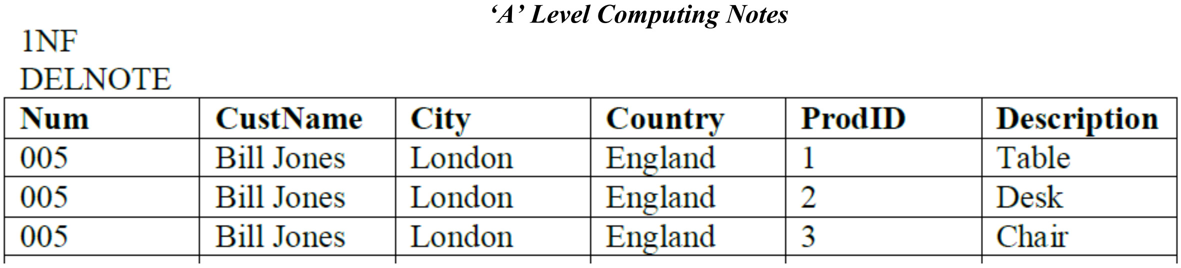
The table below is not normalised because it does not contain atomic values under the location column. It must be normalised so that it does not have repeating groups in the location column.

Num	CustName	City	Country	ProdID	Description
005	Bill Jones	London	England	1	Table
				2	Desk
				3	Chair

The above table can be illustrated as below:

Num	CustName	City	Country	ProdID	Description
005	Bill Jones	London	England	1 , 2, 3	Table, Desk, Chair

To normalise this, a new table must be created as given below:



1NF
DELNOTE

Num	CustName	City	Country	ProdID	Description
005	Bill Jones	London	England	1	Table
005	Bill Jones	London	England	2	Desk
005	Bill Jones	London	England	3	Chair

The record now has a compound key of using the Num and ProdID which is illustrated as:

DELNOTE(**Num**, CustName, City, Country, **ProdID**, Description)

Underlined fields are the keyfield which is a combination of two attributes. In this situation, we have identified a key which uniquely identifies a record.

2NF - Second normal form

The table in 1NF will lead to too much duplication of data, especially on Num, CustName, City, and Country.

- A table is in second normal form (2NF) if and only if it is in 1NF and every non-key attribute is fully dependent on the primary key. There should be no **partial dependencies**. A relation that is in 1NF and every non-primary key attribute is **fully dependent** on the primary key is in Second Normal Form (2NF). That is, all the incomplete dependencies have been removed.
- In our example, using the data supplied, CustName, City and Country depend only on Num and not on ProdID. Description only depends on ProdID, it does not depend on Num. We say that

  Num *determines* CustName, City, Country
  ProdID *determines* Description

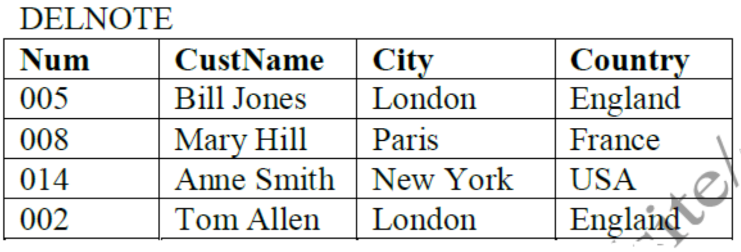
Which can be written as follows

Num → CustName, City, Country
ProdID → Description

- To solve this proble, we introduce a dummy (functional dependency) for the primary keys of the above. That is:
  Num, ProdID ⟶ *0 (dummy functional dependency/Link table)*

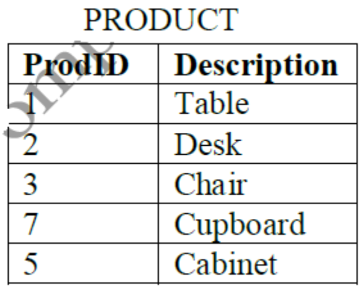
  We now get three relations, which are:

  DELNOTE(Num, CustName, City, Country)
  PRODUCT(ProdID, Description)
  DEL_PROD(Num, ProdID)
- DEL_PROD needs a compound key because a delivery note may contain several parts and similar parts may be on several delivery notes. We now have the relations in 2NF, which will appear as follows:



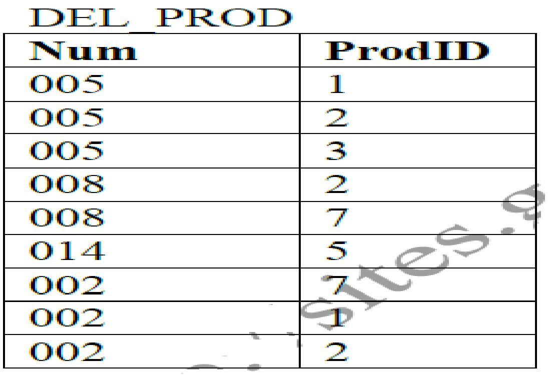
DELNOTE

Num	CustName	City	Country
005	Bill Jones	London	England
008	Mary Hill	Paris	France
014	Anne Smith	New York	USA
002	Tom Allen	London	England



PRODUCT

ProdID	Description
1	Table
2	Desk
3	Chair
7	Cupboard
5	Cabinet



DEL_PROD

Num	ProdID
005	1
005	2
005	3
008	2
008	7
014	5
002	7
002	1
002	2

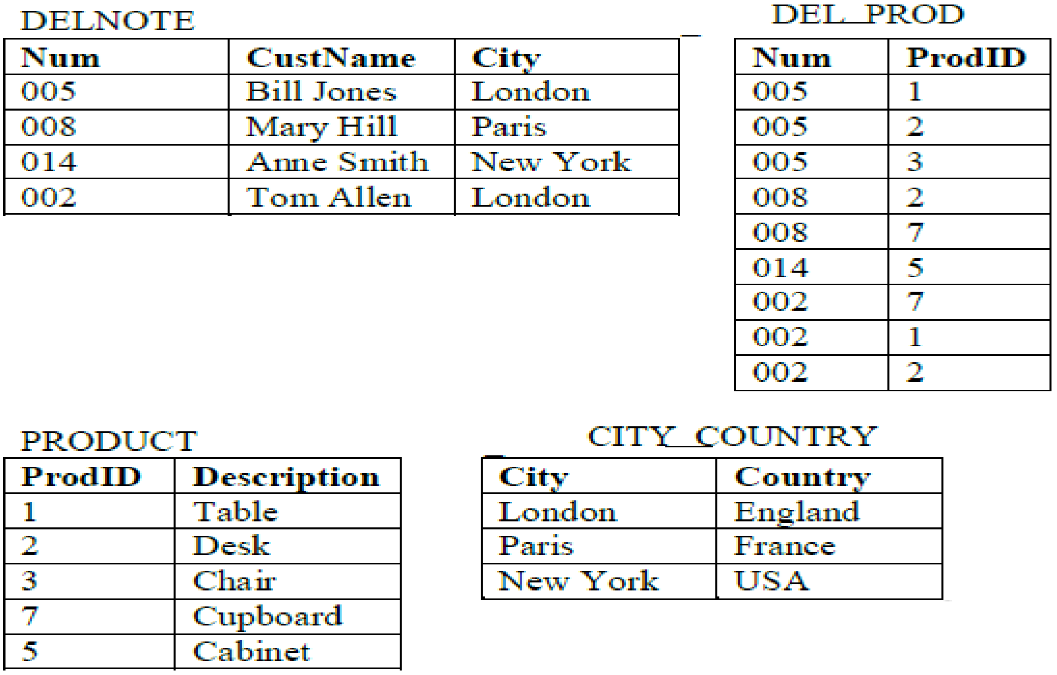
However, in the table DELNOTE, Country depends on City not directly on the primary key Num. We need to make sure that all non-key fields in all tables are fully dependent on the primary key and not on other non-key fields. We now need to normalize this into 3NF.

3NF - Third normal form

- To be in *Third Normal Form (3NF)* the relation must be in 2NF and no transitive dependencies may exist within the relation. A *transitive dependency* is when an attribute is indirectly functionally dependent on the key (that is, the dependency is through another non-key attribute).
- A relation that is in 1NF and 2NF, and in which no non-primary key attribute is transitively dependent on the primary key is in 3NF. That is, all non-key elements are fully dependent on the primary key.

  Num → City → Country (A transitive relationship: thus num transitively determines the country)
  This transitive relationship must be removed in order to have 3NF. Thus we can have the following:

  DELNOTE(Num, CustName, City)
  CITY_COUNTRY(City, Country)
  PRODUCT(ProdID, Description)
  DEL_PROD(Num, ProdID)



DELNOTE

Num	CustName	City
005	Bill Jones	London
008	Mary Hill	Paris
014	Anne Smith	New York
002	Tom Allen	London

DEL_PROD

Num	ProdID
005	1
005	2
005	3
008	2
008	7
014	5
002	7
002	1
002	2

PRODUCT

ProdID	Description
1	Table
2	Desk
3	Chair
7	Cupboard
5	Cabinet

CITY_COUNTRY

City	Country
London	England
Paris	France
New York	USA

- **BCNF – Boyce Codd Normal Form**
  - To be in *Boyce–Codd Normal Form (BCNF)* the relation must be in 3NF and every determinant must be a candidate key. A determinant is an attribute on which some other attribute is fully functionally dependent.
  - BCNF specifies that a relation should be in 3NF and should have one candidate key. In fact, a relation can be in 3NF and at the same time be in BCNF.
  - A candidate key is an attribute or combination of attributes that uniquely identify a record. A relation may have more than one candidate key. One of the **candidate keys** is chosen as a primary key and the remaining ones are called **alternate** (with no unique identification) keys.

DATABASE RELATIONSHIPS

Attributes

This is a property or characteristic of an entity. Attributes are properties of entities. In other words, entities are described in a database by a set of attributes. The following are example of attributes:

– **Brand, cost**, and **weight** are the attributes of **CELLPHONE**.

– **Student number**, **name**, and **grade** are the attributes of **STUDENT**

Entity

An *entity* is something of interest to an organisation about which data is to be held. It could be a person, place, object, event or concept about which data is to be maintained.

An entity is an object that exists and is distinguishable from other objects. In other words, the entity can be uniquely identified.

The examples of entities are:

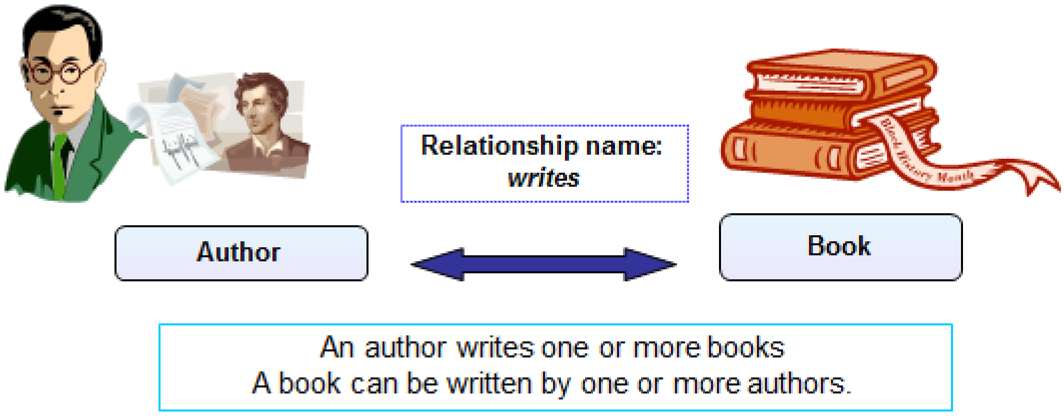
- A particular person, for example Mr Murewa is an entity.
- A particular department, for example Computer Studies Department.
- A particular place, for example Mutare City can be an entity.
- An object, e.g. stock for supermarket can be an entity.

Entity Type

An entity type or entity set is a collection of similar entities. Some examples of entity types are:
– All students at NUST, say STUDENT.
– All courses at NUST, say COURSE.
– All departments at NUST, say DEPARTMENT.
An entity may belong to more than one entity type. For example, a staff working in a particular department can pursue higher education as part-time. Hence the same person is a LECTURER at one instance and STUDENT at another instance.

Relationship

This is a link or association between entities. Relationship type is a meaningful association between entity types.
The examples of relationship types are:
– **Teaches** is the relationship type between LECTURER and STUDENT.
– **Buying** is the relationship between VENDOR and CUSTOMER.
– **Treatment** is the relationship between DOCTOR and PATIENT

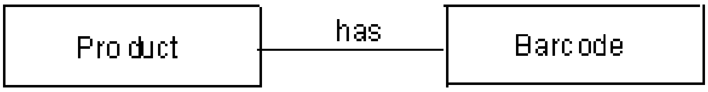


An author writes one or more books
A book can be written by one or more authors.

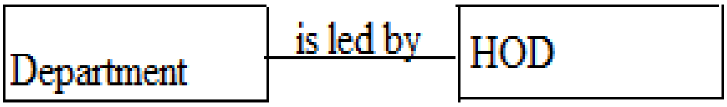
- Relationship name is an <u>active</u> or a <u>passive</u> verb.

Types of Relationship

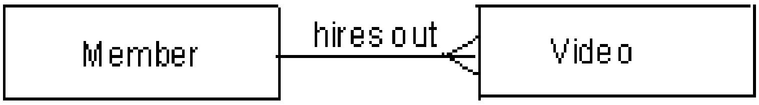
- **One-to-one**
  Eg Products in a supermarket each have a unique barcode number.



A department in school is led up by a HOD, and this person only leads one department



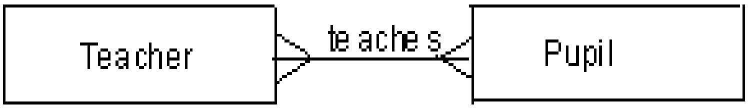
- **One-to-many**
  Eg A video club member may hire out a number of videos.



The head of department may be in charge of many staff, but these staff members only have one head of department.

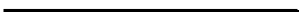
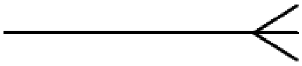
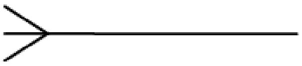
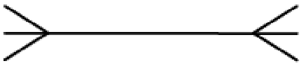
- **Many to One**
  Many videos can be hired by one member.

- **Many-to-many**
  Teachers and pupils in a school. Each teacher teaches many pupils and each pupil has many teachers.



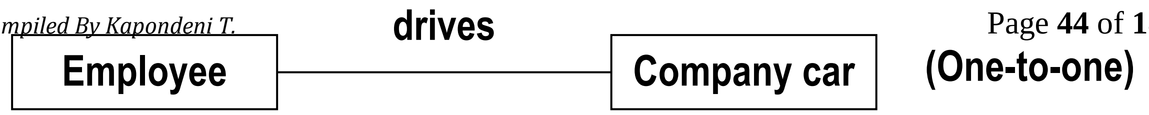
A teacher may order many books, but each book could be ordered by many teachers.

Thus in general, the relationship types are as follows:

one-to-one	represented by	
one-to-many	represented by	
many-to-one	represented by	
many-to-many	represented by	

Entity-Relationship Diagram

An ***entity-relationship*** diagram is a diagrammatic way of representing the relationships between the entities in a database.

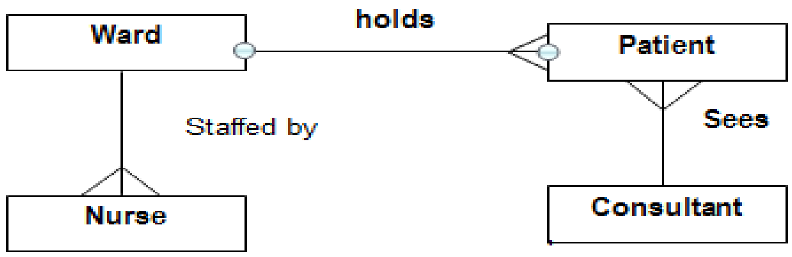


Example

- A *hospital* is organised into a **number of *wards***.
- Each *ward* has a **ward number** and a **name** recorded, along with a number of beds in that ward.
- Each *ward* is **staffed by** *nurses*.
- *Nurses* have their **staff number** and **name** recorded, and are assigned to a single ward.
- Each *patient* in the hospital has a **patient identification number**, and their **name**, **address** and **date of birth** are recorded.
- Each *patient* is under the care of a single ***consultant*** and is assigned to a single *ward*.
- Each *consultant* is responsible for a **number of *patients***. *Consultants* have their **staff number**, **name** and **specialism** recorded.

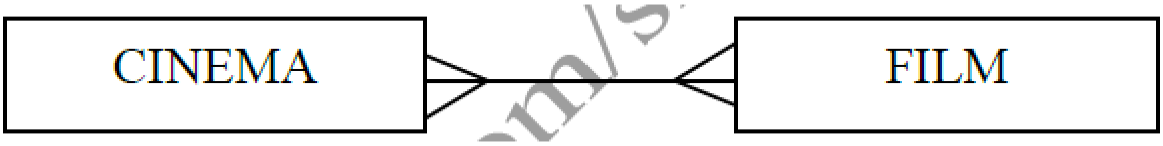
Entity Relationship Diagram

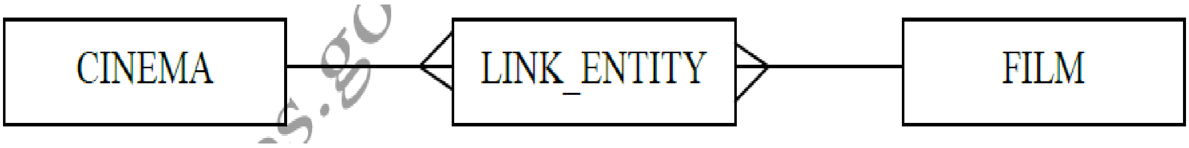
Entity	(identifier)
Ward	Ward Id
Nurse	Staff id
Patient	Patient id
Consultant	Staff id



Many – to-Many relationships are not encouraged in E-R diagrams since they violate the 3NF of databases. To remove M-N relationships, a **link entity** is used to link entities with a M-N relationship as illustrated below:

CINEMA shows many **FILMs**
FILM is shown at many **CINEMAs**





<u>Data security</u>

Refers to methods of keeping data safe from various hazards and from unauthorized access and this includes:

- Natural hazards like fire, floods, etc
- Deliberate destruction/corruption by former employees or by terrorists.
- Illegal access to data by hackers, who may steel, amend or destroy the data
- Accidental loss of data due to hardware failure, software failure, etc.

(Refer to Heathcote pages 105 - 109 for more on measures of ensuring data security. Pupils must be able to describe the following in detail:

- Keeping data secure from fraudulent or malicious damage;
- Password protection
- User IDs and passwords
- Encryption
- Access rights and user permissions
- Different user views
- Biometric measures
- Periodic backups
- Antiviruses and protection measures
- Audit trails
- System restore and Rollback facilities
- Record locking

Pupils should describe/explain concepts above.)

<u>REVIEW QUESTIONS:</u>

1. A garden design company keeps records of its customers. Each customer has had a design produced for them which will be one of a library of design types stored by the company. Each design type uses plants. Each customer is sent an account based on the number of plants in the design.

(a) Draw an E-R (entity-relationship) diagram in third normal form, based on this information. **[10]**

(b) Each delivery of plants to the garden design company is identified by a batch number. Explain how customers who received eucalyptus trees from batch 12 can be contacted. **[4]**

2. A sports club runs a number of sports teams.

Each team is made up of a number of members of the club and each member may play for more than one team. Each team has a number of coaches, but the coach's job is so time consuming that each coach can only coach one team.
Represent the above information on an entity relation (ER) diagram, in 3rd normal form, stating the primary key for each entity. **[13]**

3. (a) In relation to databases, describe what is meant by each of the following terms.

(i) Primary key. **[1]**
(ii) Secondary key. **[1]**
(iii) Foreign key. **[1]**

(b) Using, as an example, the database of student records in a school,
(i) Explain why different users should be given different access rights; **[4]**
(ii) Describe how these access rights can be implemented. **[4]**

4. A landscape garden company services a number of gardens. Each GARDEN is owned by an OWNER. Each owner may have more than one garden. Each garden has a number of PLANTS in it and each plant may be in a number of gardens.
Draw an entity relationship (E-R) diagram to represent this data model in third normal form and label the relationships. **[10]**

5. A health centre employs doctors, nurses and receptionists.
The data that is stored about the patients includes their medical history and personal information about them.
Explain the need for maintaining privacy of the data and describe methods by which the database management system (DBMS) can help to achieve this. **[6]**

6. (a) The structure of a database management system (DBMS) consists of three levels;

- External level,
- Conceptual level,
- Internal level.

State the meaning of each of these levels. **[3]**

(b) Describe the purpose of the following:

(i) the data description language (DDL), **[2]**
(ii) the data manipulation language (DML). **[2]**

7. (a) Describe the function and purpose of the following parts of a database management system (DBMS):

(i) data dictionary, **[2]**
(ii) data description language, **[2]**
(iii) data manipulation language. **[2]**

(b) Three advantages of using a relational database rather than flat files are:

(i) reduced data duplication,
(ii) improved data security,
(iii) improved data integrity.

Explain what is meant by each of these and why they are features of a relational database. **[6]**

8. Each LEAGUE has a number of TEAMs but each TEAM is only in one LEAGUE. Each TEAM plays at a number of GROUNDs during the season and each GROUND will host a number of TEAMs during the season.

(i) State the relationship between LEAGUE and TEAM.
Draw the entity-relationship (E-R) diagram to show this relationship. **[2]**

(ii) State the relationship between TEAM and GROUND.
Draw the E-R diagram to show this relationship. **[2]**

(ii) Explain how the relationship between TEAM and GROUND can be designed in third normal form. [4]

9. (a) Describe what is meant by:

(i) A backup of data [2]
(ii) An archive of data [2]

(b) The data collected by a survey team and the results of processing are both backed up and archived.

(i) Explain why it would be important to take a backup of the results of a survey [2]

(ii) Explain why it would be important to archive the results of a survey [2]

CHAPTER 3: PROGRAMMING LANGUAGES

- A **programming language** is a set of symbols in computer language that are used in coding computer programs.
- A **programming language** is a specially written code used for writing application programs e.g. C, Pascal, COBOL, BASIC, Visual Basic, C++ and Java (Originally for intelligent consumer-electronic devices (cell phones), then used for creating Web pages with *dynamic content, n*ow also used for developing large-scale enterprise applications)

- Program: a set of detailed and unambiguous instructions that instructs a computer to perform a specific task, for example, to add a set of numbers.

- Programming: A process of designing, coding and testing computer programs

- Programmer: A person who specialises in designing, coding and testing computer programs

- Problem: any question or matter involving difficulty or uncertainty and is proposed for a computer solution.

TYPES OF PROGRAMMING LANGUAGES

1. Low Level Languages (LLL):

- These are programming languages used to write programs in machine code (i.e in 1s and 0s) or in mnemonic codes.
- Low level languages are machine oriented (machine specific).
- Low level language is in two forms:

(a) Machine Language and

(b) **Assembly Language**.

a. Machine code (language)

- Is the language used to write programs in binary form (1s and 0s).
- Machine code executes without translation.
- Machine language has the following **advantages**:
  - Programs run faster since they are already in computer language. There is no need for conversion as programs are in machine language.
  - Programs occupy very small disc storage space by storing just 1s and 0s.

Disadvantages of Machine language:

- They are very difficult to learn.
- They are difficult to understand.
- Very difficult to use and therefore very few programmers use them these days.
- It takes too long to debug and therefore is prone to some errors.
- It takes too long to develop working programs.
- They are machine dependent (they can only work on type of computer designed for and not work on other computers)

b. Assembly Language:

- These are programming languages that use mnemonic codes in coding programs.
- Mnemonic codes are abbreviations used to represent instructions when coding assembly language programs, for example, LDA for Load, ADD for Addition, etc.
- One assembly language statement is equivalent to one machine code instruction and therefore programming lengthy and time consuming.
- However, assembly language programs are efficient.
- Programs also run faster as they are closer to machine language and therefore are used in designing programs that needs efficient timing, e.g. games like chess, operating systems, etc.
- Assembly language is used when there is need to access registers and memory addresses directly.
- Assembly language instructions also occupy very little disc storage space.

- Mnemonic codes are very close to machine code, hence are low level language assembly language codes.
- They however run on specific computer architecture since they are hardware aligned.
- They also contain different forms of instruction, e.g. jump, control, arithmetic, etc.
- Assembly language allows immediate, direct and other forms of memory addressing.

Application: Assembly language is used in:

- Coding operating systems
- Coding device drivers
- Coding programs for embedded systems like DVD players, decoders, etc.
- Coding encryption software

Advantages of Assembly language:

- One assembly language instruction corresponds to one machine code instruction and therefore translation is easier and faster to code.
- Programs run faster since they are close to machine code.
- They occupy very small disk storage space hence are economical to use.
- Easier for a programmer to use than machine language.

Disadvantages of Assembly Language

- They are very difficult to learn.
- They are very difficult to understand.
- Takes too long to develop working programs.
- They can be machine dependent (machine oriented) unless the machines use the same processor chip.

2. High Level Languages (HLL):

- These are programming languages that use English-like statements in coding programs, for example COBOL, Pascal, BASIC, etc.
- High Level languages are mostly used for developing user applications like stock control systems, personnel records, etc.
- There are so many high level languages because of competition from designers who want to outpace each other.
- It can also be due to the fact that we have so many application areas in real life so each high level language is designed for a specific problem (problem oriented/problem specific) to be solved in our daily lives, for example BASIC was designed for learning purposes, COBOL for business applications, FORTRAN for scientific purposes, etc.
- High Level languages are independent of the architecture of the computer.
- One statement is translated into several equivalent machine code instructions before it is executed.
- Below is an example of a BASIC program that accepts two numbers entered through the keyboard, adds them and display the result on the screen:

```
INPUT "ENTER FIRST NUMBER.", A
INPUT "ENTER SECOND NUMBER.", B
SUM = A + B
PRINT SUM
END
```

- Programs written in High Level Language are first converted to machine code before running.

High level languages have the following features:

- Problem oriented (Machine independent): they are designed to solve an application problem and therefore runs on any machine
- They are portable: they can be transferred from one machine to another and run without problem.
- Instructions are written in English statements which are easier to understand.

Facilities of High Level Languages

The following facilities of high level languages are not found in low level languages

- Selection structures
- Iteration structures
- Built-in routines to simplify input and output, e.g. INPUT, PRINT, PRINTLN, etc in BASIC.
- Built-in functions like sqr, sqrt, val, etc.
- Data structures like strings, arrays, records, etc.
- User-defined functions.

Advantages of High Level Languages:

- They are easier to understand since they are written in English-like statements which are more readable.
- They are easier to learn.
- It is easier to work with, that is to correct errors (debug) and to test programs.
- They are problem oriented and therefore can be used on any computer (not machine dependent)

Disadvantages of HLL

- Takes long to run since they need to be first converted to machine code.
- They occupy a lot of disk storage space as compared to low level languages.

Types of High Level Languages/Programming Paradigms

- Programming **paradigm** refers to methods used to categorise high level languages in terms of organising principles used by the designers. It is pattern that serves as a school of thought for programming of computers.
- Such programming paradigms are as follows:
  - Imperative/procedural languages
  - Declarative languages
  - General purpose languages
  - Special purpose languages
  - Object oriented languages (OOP)

1. Imperative /Procedural Programming languages

- These are high level programs in which the programmer specifies how the program is to be executed, following a predefined execution sequence.
- The programmer has to write instructions (in source code) that are to be strictly followed when performing a certain task.
- The programmer shows steps in the order of execution, providing a set of data.
- The order of statements is very important and is to be **obediently** followed by the computer when performing a task.
- Examples include C, Pascal, COBOL, BASIC, etc.
- Imperative languages have the following features:
  - **built-in data types** like integers, character, string, Boolean, etc
  - **user defined data types** like records, arrays, etc
  - **Declarations** for variables, arrays, constants, functions, procedures, etc.
  - **Programming statements** like assignments, control structures, procedure calls, function calls, etc.

2. Declarative Programming Languages

- These are programming languages that states **what is to be done** rather than how to do it.
- The programmer states the facts and rules associated with the problem.
- The order of stating the rules and facts is not important.
- Adding new rules, modifying existing ones and deleting some rules is very easy.

- They do not use loops, selection structures or procedures.
- Can also be called **Very High Level** programming languages or 4GL languages.
- They have the following benefits:
  - Makes it easier to program computer applications due to the following reasons:
    - Allow alternate methods of developing software, e.g. prototyping.
    - One 4GL code is equivalent to several lines of 3GL languages code.
    - Programmer productivity is increased.

Application: mostly used in designing **expert systems**.

Examples of declarative languages are **Prolog** (Programming in Logic)

3. General purpose languages

These are languages that can be used to perform a variety of takes, they are not dedicated to one particular task.

4. Special purpose languages

- These are languages that are dedicated to a specific area, for instance, for running on embedded computer systems like washing machines, mobile phones, controlling robots in a car assembly, etc.
- such languages must support real-time facilities, and have the following facilities:
  - **Real-time control facilities**: Programmer specifies times/frequency at which certain actions are to be performed, e.g. taking readings after every 3 seconds for sensors. Programmers must also program timeout if the sensor fails to take reading in the specified interval and treat this as a fault and take appropriate action.
  - **Interaction with hardware interfaces**: language must contain instructions to monitor devices like sensors and control actuators.
  - **Ability to support concurrent programming**: languages must allow two or more actions to take place at the same time.

5. Object Oriented Programming languages (OOP)

- These are programming languages that use classes and objects, allowing fatures like encapsulation, inheritance and polymorphism.
- Objects are data items with all the processing possible on the data item attached to it. (have states and methods)
- OOP languages include Visual Basic, C++, Java, Object Pascal, Common Lisp, Scheme, Perl, Python, Ruby, VB.Net and Oz etc.
- OOP languages support the following features:
  - Encapsulation
  - Inheritance and
  - Polymorphism

Identifying the state and behaviour for real-world objects is a great way to begin thinking in terms of object-oriented programming.

What Is an Object?

Objects

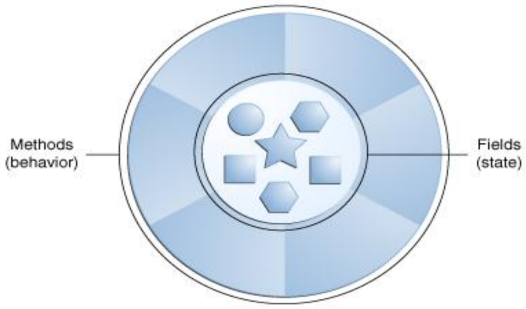
- are real world items that have **state** (attributes) and **behaviour** (operations) and belong to certain **class**, e.g. desk, car, television set, bicycle, etc.
- In general, an object is an instance of a class and is an actual real-world entity.

State:

- These are **attributes/characteristics/fields** for an object, e.g. a **car** has colour, registration number, model, etc.
- **Bicycles** have state of current gear, current speed, etc.
- Each object stores its own state.
- Dogs have state (name, colour, breed, hungry).

Behaviour (operations):

- Refers to **methods** that can be used on each object state, e.g. changing gear of a car, applying brakes, etc.
- Each class has its source code that is associated with it and defines the fields and methods.
- Dogs have behaviour (barking, fetching, wagging tail).



Class:

- A set of objects which share a common data structure and a common behaviour.
- In coding a program, a class is taken as an abstract data type that describes the fields and methods of the class.
- Each class has different access levels, which can be private, **protected or public.**
- Example of **class declarations** in **Java** are as follows:

```
class Bicycle {
        int speed = 0;
        int gear = 1;
        void changeGear(int newValue) { gear = newValue; }
        void speedUp(int increment) { speed = speed + increment; }
        void applyBrakes(int decrement) { speed = speed - decrement; }
        void printStates() {System.out.println(speed:"+speed+" gear:"+gear); }
}
```

- Bicycle is the name of the class
- Speed, gear are the states (fields) for the Bicycles class.
- ChangeGear, speedup, applyBrakes, printStates are the behaviour (methods) of the class. All bicycles have the states and operations above.
- The following coding is for the creation of objects for the class Bicycle.

```
Bicycle bike1 = new Bicycle();
Bicycle bike2 = new Bicycle();
bike1.speedUp(10);
bike1.changeGear(2);
bike1.printStates();
bike2.speedUp(10);
bike2.changeGear(2);
```

```
bike2.speedUp(10);
bike2.changeGear(3);
bike2.printStates();
}
}
```

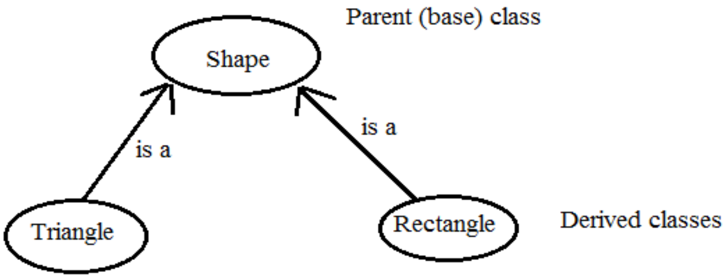
- **bike1** and **bike2** are objects of the class Bicycle.
- Their behaviour and states are the same (though they may have different values) so they belong to the same class.

Encapsulation

- This is a technique of combining operations (methods) and data (fields) into one unit as in classes.
- Encapsulation can be in two forms:
  (a) **Data encapsulation**: Hiding states internally and requiring all interaction to be performed through an object's methods. It involves restricting access to the states (fields).
  (b) **Procedural encapsulation**: users do not need to know how the behaviour happens, that is hiding operations from the user.

Inheritance

- In Object-oriented programming, **inheritance** is whereby classes can re-use (assume) commonly used state and behaviour from their parent (base) class.
- **Inheritance** is the ability of a class to use the variables and methods of a class from which the new class is derived (parent class)
- Inheritance allows a new class to be derived from an existing class.
- Inheritance therefore is a relationship among classes, where a sub-class (derived class) shares all the fields and methods of the base (parent) class, plus its own methods and states.
- Consider the **inheritance diagram** below:



- **Base Class**: This is the parent class or the first class to be created from which other class can inherit states and methods.
- **Derived class**: These are new classes that are created from the base class, and therefore have methods and states of the base class plus their own methods and states.
- The syntax for creating a subclass is use the extends keyword, followed by the name of the class to inherit from:

```
class MountainBike extends Bicycle
{
// new fields and methods defining a mountain bike
// would go here
}
```

Polymorphism

- In general, polymorphism is the ability to have the same operation performing differently in different circumstances.

- Polymorphism allows an operation to perform differently depending on the parameters that are passed.
- This is the ability of classes to use the same name in the class hierarchy for a method but each class implementing the method differently.
- In polymorphism, derived classes are able to re-define some of the base (super class) methods.

Containment/aggregation/composition

- These are links/associations between objects that allow them to communicate.
- For instance: a form on a screen is an object. On the object form, there are other objects, e.g. delete button, display button, exit button, etc.
- These buttons (which are objects) communicate with the form (another object).
- Thus the linkage between the form and the buttons is called the containment.

Event driven programming

- This is whereby the sequence of code execution is determined (triggered) by external events or by user actions, e.g. clicking a button on a form is an event that can form a program to execute certain sections of the code.
- Thus clicking a button or menu item is an event.
- This is important because the programmer cannot predict the exact sequence a user can perform his/her tasks.

Event handlers: small program codes which are invoked (called) in response to external events.
Dispatcher: small program codes that call event handlers so that events can be processed.

Advantages of OOP

Grouping code into individual software objects provides a number of benefits, including:

- **Modularity**: The source code for an object can be written and maintained independently of the source code for other objects. Once created, an object can be easily passed around inside the system.

- **Information-hiding**: By interacting only with an object's methods, the details of its internal implementation remain hidden from the outside world.

- **Code re-use**: If an object already exists, you can use that object in your program.

- **Easy debugging**: If a particular object turns out to be problematic, you can simply remove it from your application.

- **Reliability**: if codes are designed by specialists, they are more likely to free from errors due to intensive testing.

- **Time saving**: re-use of existing methods and states means less time needed to code programs.

- **Decreased maintenance costs**: programmers have les code to maintain.

- **Smaller program codes**: since the states and methods of classes are re-used, the code of the program is smaller, taking less disk storage space.

- Storage structures of an object may be altered without affecting the programs that make use of it.

Source Code and Object Code

Source Code

- Refers to the program instructions written in English-like statements (High Level Language) by the programmer, e.g. the Visual basic Code as typed by the programmer.
- Source code is human understandable language and cannot be understood by the computer.
- This therefore needs to be translated into machine code (binary) which the computer can understand.

Object Code

- Refers to a machine code version of a source code.
- All programs written in source code must be converted to object code for the computer to understand them.

TRANSLATORS

- These are programs used to convert High level Language programs as written by the programmer (source code) into machine code that the computer can execute.
- Translators are in three types, which are **assemblers**, **interpreters and compilers**, which are further explained below:

1. Assemblers:

- These are programs used to translate assembly language instructions (program) into machine language instructions before execution.
- Since each computer has its own assembly language, it also has its own assembler.
- Assembler programs are written either using assembly language, or using high level languages like C.
- Assemblers are simpler to program as compared to compilers.
- Assemblers are in two types: **One-Pass Assemblers and Two-Pass Assemblers**
  - **One-Pass Assemblers**
  - These go through the source code once and assume that all the symbols will be defined before any instruction that reference them. This is equivalent to declaring variables at the top of the source code regardless of where they are used in a high level language program.
  - It has an advantage of speed since only one phase in gone through

  - **Two-Pass Assemblers.**
  - Creates a table with all the symbols and their values in the first pass and then use the table in the second pass to generate code.
  - Has an advantage that symbols can be defined at any stage in the source code.
  - Two-pass assemblers make programs easier to read and to maintain.

Uses of Assemblers

- The uses of assemblers include:
  - They generate machine code that is equivalent to assembly language.
  - They are used to check the validity of instructions, that is, checking for syntax errors in an instruction.
  - They also assign memory locations to variables.

2. Interpreters

- These are programs that **translate** and **run** one (command) instruction of a program at a time before going to the next instruction until the end of the program, e.g. the BASIC interpreter.
- Interpreter must be present to run the program. It is used during program writing (coding) because it easily aids in debugging.
- An interpreter **translates** one instruction at a time and then **executes** it.
- The translated program retains the source code.

- They do not produce the machine code version (object code) of a program; hence translation is repeated every time the program is executed.
- If the program is run 100 times, translation for each instruction is also carried out 100 times.

Functions of Interpreters

- They translate each instruction in turn into machine language and run it.
- Allocates storage space to variables.
- They check syntax error in a program statement.
- Gives error messages to the user
- Finds wrong and reserved words of the program
- Determines wrong use of variables

Advantages of interpreters

- It is easy to find and correct syntax errors in interpreted programs.
- There is no need for lengthy recompilation each time an error is discovered.
- It is very fast to run programs for the first time.
- Allows development of small program segments that can be tested on their own without writing the whole program.
- It is easier to partially test and debug programs, especially during the programming stage.
- It is very fast to run small programs.
- individual segments can be run, allowing errors to be isolated
- running will be necessary after very minor changes
- continual compilation of whole code is wasteful/time consuming

Disadvantages of interpreters

- They are very slow in running very large programs.
- They do not produce an object code of a source code and hence difficult to use since conversion takes place every time the program is run.

3. <u>Compilers</u>

- These are programs that convert a high level language program into its machine code equivalent at one go (at once) and then run it, e.g. the COBOL compiler.
- Compiler must be present for compiling the program only and NOT during the running process.
- Creates an object code version of the source code
- Once compiled, the program no longer needs conversion since the machine code version is the one that will be run, until some changes are made to the program code.
- Compilers run faster when called and therefore may be held as library routines.
- Once compiled, the program can then be run even on a computer without the compiler since the program will already be in machine code.
- The compilation processes involves many complex stages which will be looked later in this course.

Functions of Compilers

- They check syntax errors in program statements.
- They allocate storage space to variables.
- Translate the whole program into machine code at one go.
- Run object code of the program.
- Produces a program listing which indicates position of errors in a program.
- Gives error messages to the user
- Finds wrong and reserved words of the program
- Determines wrong use of variables

Advantages of Compilers

- The object code can be saved on the disc and run when needed without the need for compilation.
- Compiled programs run faster since only the object code is run.
- The object code can run on any computer, even those without the compiler. Therefore compiled programs can be distributed to many users and used without any problems.
- The object code is more secure since can cannot be read without the need for reverse engineering.
- Compilers indicate the line numbers with syntax errors and therefore assist programmers in debugging programs.
- They are appropriate even for very large programs.

Disadvantages of Compilers

- Slower than interpreters for running programs for the first time.
- They can cause the computer to crash.
- Difficult to find errors in compiled program.
- There is need for lengthy recompilation each time an error is discovered.

Difference between High Level Languages and Low Level Languages

	High Level Language	Low Level Language
1	Written in English like statements	Written in 1s and 0s (machine code) or in mnemonic codes.
2	Easier to work with	Difficult to work with
3	Easier to understand	Difficult to understand
4	Are problem oriented and can be used on any computer	Machine oriented
5	Slower in execution since they need to be first converted to machine code before running	Faster in execution since they are in machine code already.
6	Occupy large disk storage space on the computer	Occupy small disk storage space on the computer
7	They are machine independent	They are machine dependent

3. Very High Level Languages (VHLL) – 4GLs:

- These only specify the desired end result and do not indicate steps that the computer needs to take to make the calculation, e.g. SQL used in relational databases.
- 4GLs are flexible and easy to use since no coding is required.

4. Natural Language – Artificial Intelligence and expert systems:

These are programs that mimic human reasoning and learn from experiences.

Programming Language Generations

1. First Generation Languages (1951-58) – Machine language: These used binary form to code programs. This is the Machine language version. Programming was tedious to do.

2. Second Generation Languages (1959-64) – Assembly Language: Assemblers, compilers and interpreter became available to represent machine code. This generation saw the use of assembly language. Programming was simpler and less tedious.

3. Third Generation Languages (1965-70)- High Level Languages: This saw the development of High Level Languages like BASIC, COBOL, FORTRAN, etc

4. Fourth Generation Languages (1971) – Very High Level Languages (4GLs): This saw the development of non-procedural languages like SQL, PARADOX, etc.

5. Fifth Generation Languages(1981) - Natural Language, artificial intelligence, expert systems like PROLOG, LISP.

Features of High Level Programming Languages

1. **Programming Constructs**

   These are the basis from which high level languages are built. Programming constructs includes:

   (a) Control Structures

   (i) If – Then – Else Construct

   (ii) Case statement

   (b) Looping Structures

   (i) For – Next construct

   (ii) Repeat – Until construct

   (iii) While – Endwhile

2. **Operators**

   Operators are used to manipulate data and they can be

   (a) Arithmetic operators

   e.g. b+c-d*e/f

   (b) Logical operators

   And, or, Not

   (c) Assignment operators

   =

   (d) Comparison/relational operators

   >, <, <=, >=, <>, Is

3. **Identifiers**

   - An identifier is a unique label of a data item of element of a program and comprises of one or more characters.
   - Identifiers includes variable names, procedure names, constants, etc. Identifiers must not be reserved words.
   - Identifiers can be user defined as long as they are not reserved words.

4. **Constants**

   - A constant is a data item (variable) whose value does not change during program execution.
   - Its value is fixed.
   - Constants are used to represent data items with fixed values, e.g. the value of **pi**. In VB 6.0, a constant is declared as shown below:

```
Public Sub compute_interest()
        Const pi=3.14159
        Dim rs As New Recordset
        Dim rs_loanpay As New Recordset
        ……………..
        ……………….
        ……………..
End Sub
```

If the constant is a string value, it must be enclosed in quotes, e.g.

Const Name = "Kapondeni"

In this case, the value of name will never change (when Name is declared as a constant)

5. Variables

- A variable is a name given to a memory location that stores a certain value, which may change (the value) during program execution.
- Variables can be field identifiers, e.g. *surname* is a valid variable name.
- Variables must not be reserved words.
- Variables must be unique in the procedure or program (if all are global).
- Variables are declared at the beginning or at some point inside the program code. Every variable must be declared before use, otherwise an error is generated.
- Variable names, **as are all identifiers**, start with an alphabetic character.
- They can be one character or a string of characters.
- Variable names can be alphanumeric (combination of alphabetic and numbers).
- They must be **one word** and must be related to the data stored in them so that the programmer cannot be confused, e.g. Surname should be variable that stores a surname.
- If two words are used as a variable, they must be joined by and underscore (_), with no spaces between the words, e.g. *Student_Surname*, **NOT** *Student-Surname*.
- Alternatively, one may join the words as follows, ***StudentSurname.***
- Variable names should not be too long, 8 characters are ok although VB supports longer variables names but must not be more than 255 characters.
- Variables can store numeric, character or string values and must be declared appropriately.
- In Visual Basic 6.0, variables must be declared first before they are used. The keyword ***Dim*** is used to declare variables, and each variable should have a data type, e.g.

```
Dim number As Integer
Dim Name1 As String
Dim x, Number_1, AmountOwed As Integer
```

- The scope of variables can be **global** or **local**.
- Scope of a variable refers to the extent (within the program) that the variable is recognised (used), especially in relation to other modules and functions within the program.
- Constants can also be declared as global or local.
- In Microsoft Visual Basic variables have the following features:
  - ✓ Must start with an alphabetic letter as the first character.
  - ✓ Must be one word, no spaces allowed
  - ✓ Must not contain a period (**.**), exclamation mark (**!**), or the characters **@**, **&**, **$**, **#** in the name.
  - ✓ can't exceed 255 characters in length.
  - ✓ Must not be reserved words
  - ✓ Must be unique. You can't repeat the same variable name within the same level of scope. For example, you can't declare two variables named, age in the same procedure

a. Global variables

- These are variables that are accessed and can be used by any procedure or function within the same program.
- They are public variables
- The value of the variable exists throughout the program.
- Global variables are declared outside the procedure.
- In VB 6.0, global variables are declared as follows:

```
Public Sname As String
```

The word Public implies that it is a global variable.

b. Local variables

- These are variables that are defined within a procedure and that are accessible just within the procedure they are declared.
- They are defined within the procedure.

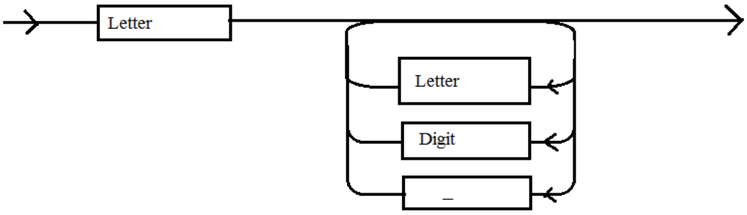
- They are private variables
- The value of the variable only exists within the procedure it is defined.
- They are therefore local to that procedure in which they are declared and therefore cannot be used (not accessible) by other procedures.
- Local variables are declared as follows:

  Private Sname As String

Form Variables

Syntax diagram of a variable:

- Syntax diagrams are graphical representations of the structure of valid variables.
- They convey what is legal for variable(or identifiers) in a certain programming language as shown below:



- The above diagram shows that every variable name starts with a Letter, followed by any of the following (Letter, digit or _) at any position, or a mixture of both in any order, as long as the first character is a letter.
- Using the diagram above, a variable like ***3_Name,*** is invalid since it starts with a number.
- variable names must follow the rules of the language
- the translator tries the rules against the variable names used and reports any errors
- The contents of variables must be of a specific type otherwise an error created by the attempted use of anything else.

6. Reserved words

- Reserved (key) words are **identifiers** with a **pre-defined meaning** in a **specific** programming language, for example ***Dim***, ***if, End, integer,*** **As**, etc. in Visual basic.
- Reserved words must **not** be used as variables.
- Each programming language has its own reserved words, which may differ from other languages.
- translator program maintains a dictionary of reserved words
- if the reserved word used is not in this dictionary then an error has been made and message may be given which suggests one close to spelling provided

7. Expressions

- An expression is a construct made up of variables and operators that makes up a complete unit of execution.

  Example:

  *NumberA = a+b-c*d*
- The above is a statement. However, ***a+b-c*d*** is an expression found in a statement.
- Expressions can be arithmetic (as given above), Boolean or string expressions. For example,

  (a>b) and (a>=c)
- Operator precedence is very important in evaluating expressions and therefore it is important to enclose expressions is brackets where possible. Operator precedence is as follows, starting from the highest to the lowest:

  ( ), Not, ^, {*,/,}\,Mod, {+, -,}{=,<,>,<=, >= },…

***NB**: Operators is set braces indicates that they are in the same level.*

- Arithmetic expressions are evaluated first, followed by comparisons and lastly logical expressions.

8. Statements

- A statement is a single instruction in a program which can be converted into machine code and executed.
- A statement can just be one line of program code but in some cases a statement may have more than one line.
- For example: *Name = "Marian"* is a statement.

Example 1

```
NumberA = a + b
```

- this is an assignment statement, that is, variable NumberA is assigned the sum of the values of variables a and b. thus if a=2 and b = 3, NumberA is assigned the value 5.
- An assignment is an instruction in a program that places a value into a variable, e.g total = a + b
- The above is just one line statement.

Example 2

```
If a>b Then MsgBox "a is bigger than b.", vbExclamation
```

- The above is a one line statement composed of if statement.

Example 3

```
  If b < 0 Then
   MsgBox "b is less than zero. Command cannot be executed", vbExclamation
   Exit Sub
End If
```

- This is a statement (starting at the first if and ending at End If.
- This state comprises of other statements between it.

9. Block structure

- A block is a group of zero or more statements between balanced braces and can be used anywhere a single statement is allowed. For instance

```
if (condition) Then ( begin block 1)
         ……………….. (end block 1)
else (begin block 2)
         ………………..
End If (end block 2)
```

10. Functions

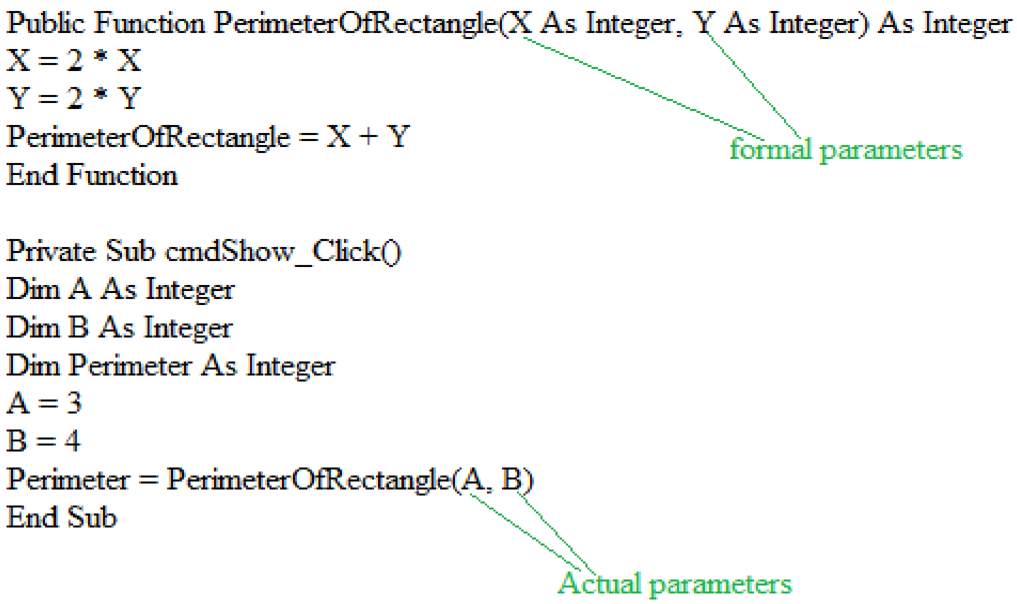
- A function is a self-contained module that returns a value to the part of the program which calls it every time it is called/executed.
- A function can be just an expression that returns a value when it is called.
- A function performs a single and special task, e.g. generate a student number.
- Because they return a value, functions are data types, e.g. integer, real, etc.
- Functions can be **in-built** functions or **user-defined** functions.
- In-built functions are pre-defined procedures of a programming language that returns a value, e.g. **Val** (returns a numeric value of a string), **MsgBox** (creates a textbox on the screen), **Abs** (returns an absolute value of a number), etc.
- Visual Basic has in-built date functions, string functions, conversion functions, etc.
- A user-defined function is a procedure (module) that returns a value whenever it is called. The structure of a user defined function is as follows:

```
Public Function count_rec(ByVal rs As Recordset) As Boolean
        If rs.RecordCount <= 0 Then
                MsgBox "There is no record in the table.", vbExclamation
                count_rec = True
        Else
                count_rec = False
        End If
End Function
```

- Note that a Function starts with the word Function and ends with the statement End Function. This function returns a Boolean value(either true or false). The function name as just after the word Function, i.e count-rec in the case above.

11. Procedures

- A self-contained module that does not return a value.
- Procedures usually starts with the key word Procedure and then procedure name, e.g *Procedure FindTotal*. Procedures are user defined.
- The name of the procedure should be related to its task
- Each procedure name must be unique within the same program.
- A procedure can be called from the main program or by other modules.
- A procedure is called by stating it name
- Parameters are usually passed when calling procedures.
- **Parameters/arguments** are values that are passed from one procedure to another and can be the actual values or variable names. They are therefore values given to a function by statements from other modules.
- Parameters can be **formal** or **actual** parameters
  Actual parameters: these are arguments found in the calling module/statement, could be variables or actual data like 30, 40.
  Formal parameters are those variables that receive data from calling module or statement.



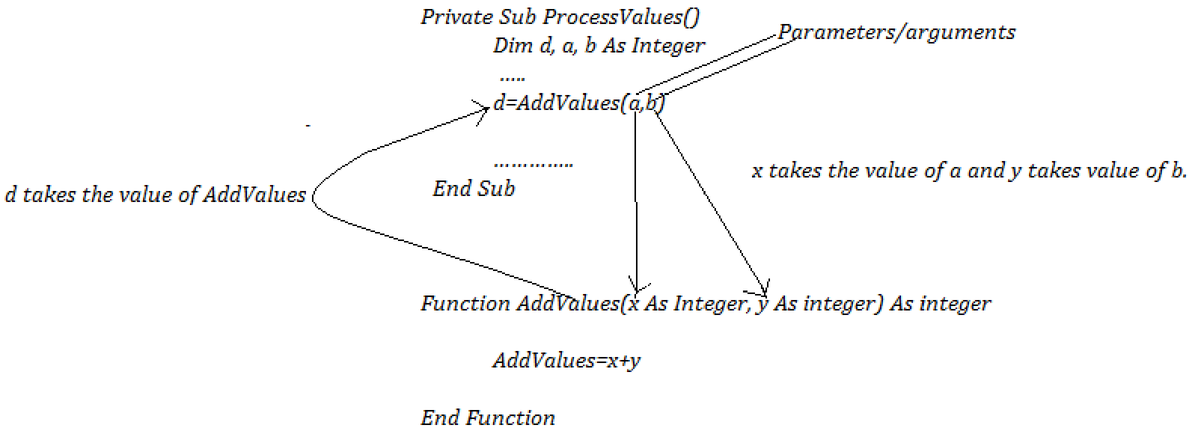
```
Public Function PerimeterOfRectangle(X As Integer, Y As Integer) As Integer
X = 2 * X
Y = 2 * Y
PerimeterOfRectangle = X + Y
End Function

Private Sub cmdShow_Click()
Dim A As Integer
Dim B As Integer
Dim Perimeter As Integer
A = 3
B = 4
Perimeter = PerimeterOfRectangle(A, B)
End Sub
```

- When processing of the called procedure is finished, processing goes to the next stated after the calling statement.
- Parameters can be passed by **value** or by **reference**

Passing Parameters by Value

- Passing by value is whereby the actual value (or value of a variable) is searched (e.g 20), copied and passed to another module.
- In this case, variables whose values are passed will NOT be altered even if the values of variables in the calling procedure or function change.
- Thus only the values of the variables are passed, not the variable itself.
- A copy of the value of the variable is passed and not the variable itself.



```
Private Sub ProcessValues()
        Dim d, a, b As Integer
        .....
        d=AddValues(a,b)

        .............
    End Sub

Function AddValues(x As Integer, y As integer) As integer

        AddValues=x+y

End Function
```

Parameters/arguments
d takes the value of AddValues
x takes the value of a and y takes value of b.

Passing Parameters by reference

- Passing by reference is whereby the address of the value is passed, allowing it to be altered if necessary.
- This is when calling procedure passes the variable to another function, whose processes may alter the value of the variable in the calling procedure.
- In this case, the original variable's memory is addressed and can therefore be altered by the called function. E.g.

```
    Private Sub ProcessValues()
            Dim d, a, b As Integer

           a = 2
           b = 3
           d = AddValues(a,b)
            .............
     End Sub


  Function AddValues(ByRef x As Integer, ByRef y As Integer)
       x  = 20
       y = 30
       AddValues= x + y
End Function
```

- If the programmers does not specify **ByVal** or **ByRef** function, Visual Basic assumes that it if **ByRef** by default.

12. Semantics

The meaning attached to statements and the way they are used in a certain language.

13. Syntax

- These are grammatical rules and regulations governing sentence construction and layout of different programming languages.
- For example, Pascal uses a semi-colon(;) at the end of each instruction, it also uses the reserved (key) word *writeln* to display items on the screen, etc.
- Each programming language has its own syntax.
- A program with syntax errors does not run.

14. Literals

- A literal is a variable which is given a fixed value within the code of the program.
- A literal is the source code representation of a fixed value.
- Literals are represented directly in your code without requiring computation, as shown:

Dim result As Integer
Dim Name As String
Result =20

Name = "Kapondeni"

15. **Input operations**

    This Refers to statements that prompts the user to enter data into the computer. E.g.

    ID = InputBox("Enter Member Data to Search")

The above displays an input box with the message in bracket that prompts the user to enter data that needs to be searched.

16. **Output operations**

    Refers to statements that allows display or printing of results on the screen or to the printer, e.g.

    frmPayments.Show

    allows, the payment form to be displayed on the screen.

    MsgBox ("Record not Found")

    Displays a message box with the message in brackets.

17. **File handling operations**

    Allows user to manipulate files. E.g. in Visual basic:

    rs.Close
    While Not .EOF

DATA TYPES

- Data types describe the nature of data handled by programs.
- These are important as they enhance program readability and maintenance.
- Data types can be simple (integers, string, etc.) or complex (arrays, lists, records, etc.).
- Data types are declared at the beginning or at some point inside the program code.
- Data types for variables must be properly declared.

a. Numeric data types

1. **Integer**
   - Used to represent whole numbers, positive or negative, and occupy 2 bytes of memory.
   - Decimal values will be rounded to the nearest whole number.
   - Values accepted range from -32 768 to + 32 767. Default value is 0.
2. **Byte**
   - Stored in a single **8-bit** unsigned data value ranging from 0-255.
   - Used to store binary data.
   - Default value is a 0.
3. **Real**
   - A data type that is used to represent values (numbers) with decimal point values, e.g 23.56 is a real number.
   - Can be used to represent averages.
   - Some languages take this to be **float** data type.
4. **Currency**
   - Occupy 8 bytes with 0 as default value.
   - Used for handling monetary values and support up to 4 digits to the left and 15 digits to the left of the decimal point.
5. **Date**

   Represent date and time values and occupy 8 bytes.

6. **Double**
   - Stores a double precision floating point number with decimal places but longer than single.
   - Default is 0 and occupies 8 bytes
7. **Single**
   - Stores a single precision floating point number, with decimal places.
   - Default is 0 and occupies 4 bytes.

8. **Long**
   - Short for long integer and default value is 0.
   - It occupies 4 bytes from -2 147 483 648 to -2 147 483 647.

b. String data types

1. String

- A string is data type that stores one or more characters, which can be alphabetic or alphanumeric.
- If a variable is declared as a string, it must not be used for daily life mathematical calculations because strings are not numbers.
- Strings are declared as follows:

Dim Name As String

- If a value of string variable is assigned within the program code, it must be enclosed in quoates e.g.

Name = "Tungamirai"
Name = ""

- The second example will be a null (un-initialised) string. It is different from 0 or space but is distinct in nature.
- Strings can be of **fixed length** or **variable length**

Fixed length: has specified maximum number of characters. E.g.

*Dim Sname as string * 20*

- The field will be allocated 20 spaces in the computer memory.
- If a name shorter than 20 characters is entered, blank spaces will remain unoccupied since they would have been assigned to *Sname* already.
- If too long than allocated space, the extra characters are cut.

Variable length:
Does not specify the number of characters, each data items occupies the number of spaces it requires.
This is declared as follows

Dim Sname as string

2. **Boolean**
   - Represents instructions that evaluate to either **True** or **False** only. The default value is **False** and occupy 2 bytes of storage.
   - Mostly used together with ***If….. Then……. Else*** Construct.
3. **Character**
   - A data type that holds just one alphabetic or special character like %, $, a, etc.
   - can be written as ***char*** in most programming languages.

User-defined data.
These are defined by the user depending on the method of solution, e.g classes when the used define own classes.

Enumerated data:
These are data types with a list of items that are pre-defined, e.g days of the week, months of the year, etc.

Word: a computer word is group of bits that can be handled or transferred by the processor as a single unit (word length).

Factors to consider when choosing a programming language

- Nature of the application.
- Availability of suitable compilers/interpreters.
- Availability of needed facilities in the programming language for system implementation.
- Availability of compatible hardware.
- Availability of expertise of the programmers.

TOP-DOWN PROGRAM DESIGN (Modularisation)

- This refers to the splitting of a program into simpler subtasks called modules which will be easier to solve.
- For example, a program can be split into modules to **Accept Number, Add, Divide, Subtract and to Display Results**.
- The main program then will have calls to the modules, which may also call other modules.
- Modules are also called procedures, routines, sub-routines or functions.
- The splitting of a problem into a series of self-contained modules is called modularisation (modular programming).
- However, there is actually a difference between a procedure and a function.

Advantages of modularisation (modular programming)

- Minimises coding since a module can be called many times from different modules using parameters can be used to pass data within the program.
- Programmer can concentrate at one simpler task at a time.
- Modules are simpler and easier to understand and to solve.
- Modules are easier to test and to debug and therefore lessening testing time.
- Program modification is easier since changes are isolated with specific modules.
- Many programmers are employed, one on each of the modules.
- More experienced programmers can be assigned complex modules.
- It saves programming time by sharing tasks.
- A large project will be easier to monitor and control.
- It is easier to update (modify) modules than the whole program at larger.
- Fewer bugs since each set of programming commands is shorter
- Allows library programs to be inserted

Disadvantages of Modularisation

- Documentation will be long and thorough, therefore may take time to produce
- Can lead to problems of variable names as the modules are developed separately.
- However, it may be difficult to link the modules together.

***NB: Library programs:** this refers to a collection of standard pre-written programs and subroutines that are stored and available for immediate use by other modules in the system.

Library programs are programs that are required by other modules during execution, e.g. the Dynamic Link Libraries (DLL) in the Windows environment.

Libraries contain common tasks like saving, deleting, etc.

Library programs are referenced by most modules in the systems and they provide data to other modules.

***NB: Stepwise refinement**: a technique used in developing the internal working of a module.

Aims of program design

- Program designing must produce a program with less maintenance and debugging costs later on.
- Therefore the aims are to produce a program with the following features:
  - **Readability**: program must be easy for programmers to read and understand
  - **Reliability**: program must always perform what is was designed to do.
  - **Maintainability**: program must be very easy to modify or change when the need arises.

- **Performance**: the program must be efficient and fast in performing whatever it was designed to perform.
- **Storage saving**: program must occupy as little storage space as possible.

BACKUS NAUR FORM AND SYNTAX DIAGRAMS

- These are methods of specifying a set of rules that specify precisely every part of the language.
- Backus Naur form can be used to define/expressed the structure of integers, variables, etc.
- For instance

An integer is a sequence of the digits 0, 1, 2, … , 9. Thus the following are all valid integers.

0
2
415
3040513002976
0000000123

- Thus, an integer can be a single digit. We can write this in Backus Naur form as

<integer> ::= <digit>

This is read as 'an integer is defined to be (::=) a digit'.

But we must now define a digit. A digit is 0 or 1 or 2 or … or 9 and we write this as

<digit> ::= 0|1|2|3|4|5|6|7|8|9

- The vertical lines means OR
- Thus our full definition of a **single digit integer** in Backus Naur Form is

<integer> ::= <digit>
<digit> ::= 0|1|2|3|4|5|6|7|8|9

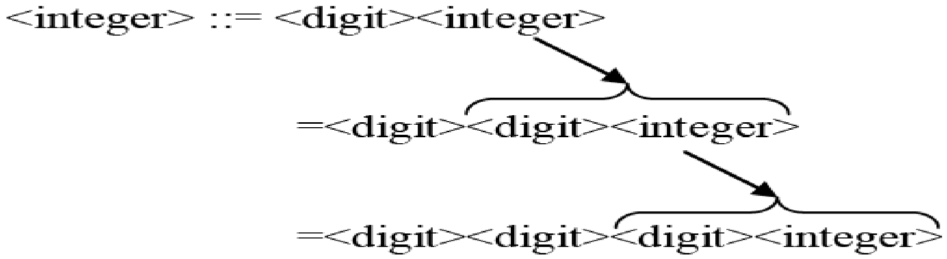
Specifying integers of any length.

Let's take our integer as 589, and see how we can define it in Backus Naur Form.

An indefinitely long integer is defined as

<integer> ::= <digit><integer>

This is a recursive definition as integer is defined in terms of itself. Applying this definition several times produces the sequence



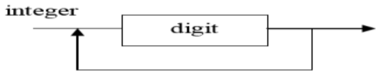
To stop this we use the fact that, eventually, <integer> is a single digit and write

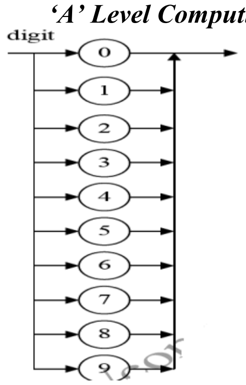
<integer> ::= <digit>|<digit><integer>

We now have the full definition of an unsigned integer which, in BNF, is

<unsigned integer> ::= <digit>|<digit><unsigned integer>
<digit> ::= 0|1|2|3|4|5|6|7|8|9

The above can be shown using a syntax diagram as:





Signed integers in BNF

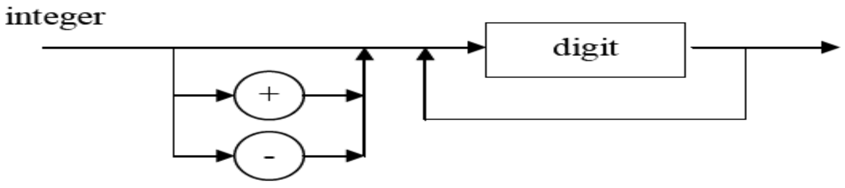
Signed integers includes +78, -90, etc.
This is simply an unsigned integer preceded by a + or – sign. Thus

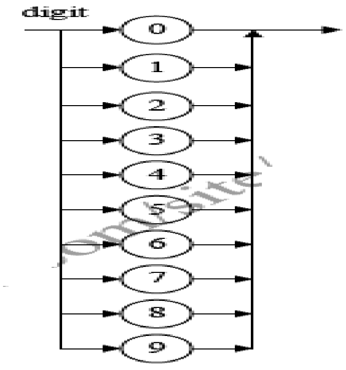
<signed integer> ::= + <unsigned integer>| - <unsigned integer>

Thus the final answer will be as

<integer> ::= <unsigned integer>|<signed integer>
<signed integer> ::= + <unsigned integer>| - <unsigned integer>
<unsigned integer> ::= <digit>|<digit><unsigned integer>
<digit> ::= 0|1|2|3|4|5|6|7|8|9

Using syntax diagrams, this can be illustrated as:



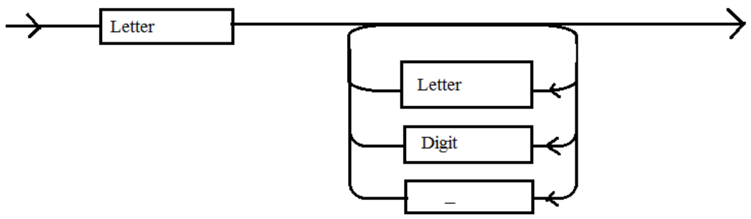


Variables in BNF

Valid variables start with a letter (Upper or lowercase) and followed by any character (which must be a letter, digit or underscore) upt to any length. This can be defined in BNF as:

<variable> ::= <letter>|<variable><character>
<character> ::= <letter>|<digit>|<under-score>
<letter> ::= <uppercase>|<lowercase>
<uppercase> ::= A|B|C|D|E|F|G|H|I|J|K|ZL|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<lowercase> ::= a|b|c|d|e|f|g|h|i|j|k|zl|m|n|o|p|q|r|s|t|u|v|w|x|y|z
<digit> ::= 0|1|2|3|4|5|6|7|8|9
<under-score> ::= _

The above can be illustrated in a syntax diagram as:



The Compilation Process

The compilation process takes a number of stages, which are:

- Lexical analysis
- Syntax analysis
- Code generation
- Code optimisation

(a) Lexical analysis

- The first stage of compilation in which the source code is broken into logical smaller chunks called tokens.
- This process is done by a program called a lexical analyser.
- A token is the smallest independent part of a program which has some meaning, e.g. variable.
- Lexical analyser also preforms the following:
    - Removes redundant parts of source code, e.g. spaces/white space
    - Removes comments from the program
    - Checked validity of reserved words within the program code
    - Tokenises reserved words (replacing of reserved words like If, While, etc. with shorter tokens)
    - Tokenises operators
    - Checks validity of symbols and variable names variable names
    - Creates the symbol table

(b) Syntax analysis

- The second stage which involves determining if the string of input tokens parsed from the lexical analysis stage forms valid sentences and if they conform to the grammar of the language.
- Some of the activities involved are:
    - Checking of grammar of statements to determine if they conform with the language
    - Issuing of error diagnostics (methods/suggestions for solving errors)
    - Checking for existence of jump destinations (presents of called modules)
    - Control constructs checked to determine if they are valid
    - Checking that variables used in the program have been declared
    - Checking for existence of library modules
    - Checking if expressions are valid, e.g. if the correct number of brackets are used.
    - Determining the priorities of operators in an expression
    - Generates a dictionary, which is a list of variables used in the program and their data types.
    - Produces intermediate code

This stage can also be called **parsing** and is done by a program called a **parser**.

(c) Code generation

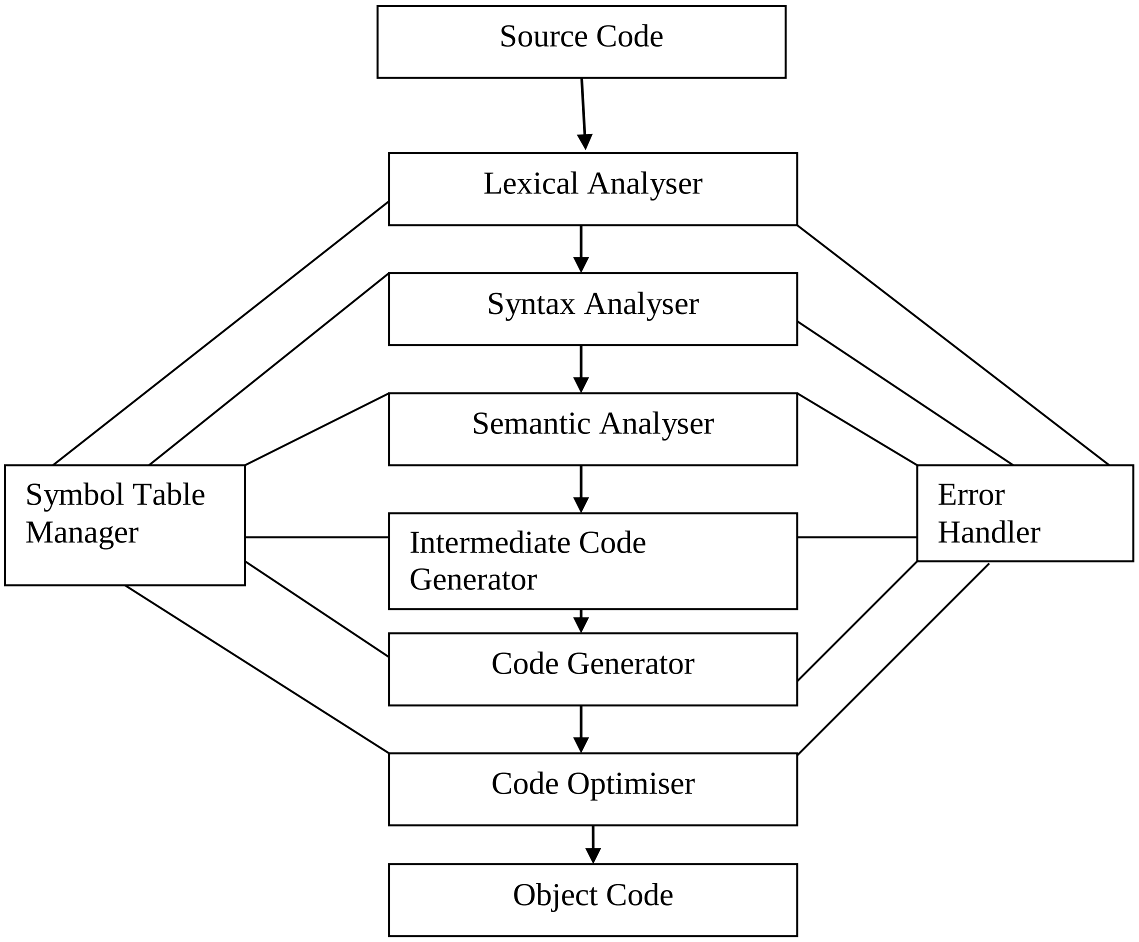
- This is a compilation stage where the code specific to the target machine is generated, i.e. the machine code.
- It involves the production of a machine code program/intermediate code which will produce the results intended by the source code.

- The following are also involved in code generation:
  - Intermediate code produced which can then be turned into executable code/machine code
  - the production of a machine code program
  - Address of variables calculated and stored in symbol table

(d) Code optimisation

- In this stage, the machine code is optimised which involves using rules to make code as small and efficient as possible by removing any duplicate or redundant instruction which improves speed of execution

In summary, the stages of compilation are illustrated as follows:



<u>Analysis Phases</u>

Consists of the lexical analyzer, syntax analyzer and the intermediate code generator.

Lexical Analysis

This is where scanning of the source program is done. Each sequence of characters that have an atomic meaning are recognized and represented by a token.

A token represents a class of valid letters, for example a program is going to analyse numbers, variables and functions.

Syntax Analysis

This is the stage where tokens are grouped into large structures, for example, assignment statements.

Semantic Analysis

This is where there is transition of tokens into code generation and complex errors are detected.

Intermediate Code Generator

This is where compilation from the source language into a target program.

Synthesis Phase

This phase consists of the Code Optimiser and Code Generator.

Code Generation

The target code is generated from the intermediate code to perform static scheduling and register allocation.

Code Optimiser

There is reduction of the number of instructions in order to allow the program to run faster.

Compiler Dealings with variables

During lexical analysis

- Characters in the variable name are tokenised
- Variable name is added to symbol table…
- Data type added…
- scope is added/block(s) in which variable is valid

During syntax analysis

- variable checked against syntax of the language, e.g. syntax diagram.
- Variable names which do not match the rules are reported in error diagnostics
- Statements containing variables are checked for syntax
- Position in table is hashed from the name
- Variable declarations are checked/also variable use

During code generation

- Address of variable calculated
- added to symbol table

Compiler Dealings with syntax errors

- Reserved word is isolated
- Computer keeps list of all reserved words in the dictionary.
- if not in list of reserved words, then an error message is conveyed
- If reserved word identified then syntax table is checked for expected form of statement.
- If form of statement does not match an error issued
- Variable names checked against rules for variable names
- Variable declarations are checked to determine if they are present.

Loading programs and linking modules

Loaders

A loader is a program that loads compiled program/modules into the computer's memory. Some functions of loaders are:

- Loads the individual modules into the computer's memory
- Loader decides where modules are to be placed in memory
- Adjusts memory addresses for individual module according to where they are placed
- loads library routines
- Copies program from storage location into memory

Loaders are in two forms, **absolute** and **relocating** loaders

(a) Absolute loader: refers to a program that loads program into a fixed area of memory each time a program is run. The loaded program can only work when it is loaded to that single address in memory.

(b) Relocating Loader: This refers to a program that loads a program anywhere in the memory and the loaded program work without any problem. Thus address of a loaded program is recalculated every time the program is loaded into memory.

Also relocating can be in two forms:

Static relocating: Once the program is loaded into a memory address, its relocatability is lost, that is, it cannot be moved to any other address during execution.

Dynamic relocating: the program can even be moved to any other address during execution.

Linkers

A linker is a program that compiles all loaded modules and then create linkages between them. Other functions of linkers are:

- Linker joins the modules compiled correctly
- Calculates addresses of the separate modules
- Allows library routines to be linked to several programs
- Ensures jump instruction from module to module properly addressed
- Produces an executable file
- matches up address references between modules

Editors

These are programs that are used to key–in and amend source code. It is also used to display and edit text before compilation.

REVIEW QUESTIONS

1. Explain the terms

(i) data encapsulation
(ii) inheritance
(iii) Polymorphism

when applied to programming in an object oriented programming language. (4)

2. Distinguish between procedural languages and declarative languages. (4)

3. Explain the passing of parameters by reference and by value. (4)

4. (a) Explain the difference between the translation techniques of interpretation and compilation [2]

(b) Give two advantages of each of the two translation techniques. [4]

5. An amount of money can be defined as

• A $ sign followed by either
• A positive integer or
• A positive integer, a point, and a two digit number or
• A point and a two digit number

A positive integer has been defined as <INTEGER>
A digit is defined as <DIGIT>::= 0/1/2/3/4/5/6/7/8/9.

a) Define, using Backus Naur form, the variable <AMOUNT OF MONEY>

b) Using the previously defined values of INTEGER and DIGIT, draw a syntax diagram to define AMOUNT OF MONEY.

6. State the three stages of compilation and briefly describe the purpose of each. [6]

7. Explain in detail, the stage of lexical analysis. [6]

8. Explain the role of

(i) linkers,
(ii) loaders

in the running of programs [4]

9. Two of the stages which a high level language program undergoes during compilation are lexical analysis and syntax analysis.

Discuss how errors are discovered during each of these two stages. [5]

10. (a) (i) Describe what is meant by source code. [2]

(ii) Explain why source code needs to be translated into object code. [2]

(b) State what is meant by the following types of programming error:

(i) syntax error [1]

(ii) arithmetic error [1]

11. (a) Explain how the translator program prepares the programmer's code into a program that the machine can run. [2]

(b) (i) Explain what is meant by a procedure. [2]

(ii) Describe how procedures and the programming construct "selection" can be used to code a simple menu system for a user. [3]

12. (a) Explain the difference between interpretation and compilation of a program written in a high level language. [2]

(b) Explain what happens during the lexical analysis stage of compilation. [5]

(c) Describe two things that happen during code generation. [4]

13. (a) Programs can be designed in modular form.

Discuss the advantages and disadvantages of designing programs in modular form. [5]

(b) A program is to be written which will update the records in a sequential file and then produce a backup copy.

Describe, using a diagram, the way that this problem can be split into modules to prepare it for coding. [5]

14.(a) Explain why a program, written in a high level language, needs to be translated before it is run on a computer. [2]

(b) Describe the difference between interpretation and the compilation of a high level language program. [4]

(c) Explain how errors in the

(i) reserved words,

(ii) variables

Used in high level language instructions are recognised by the translator program. [4]

15. Describe each of the following programming paradigms

(i) Object-oriented, [2]

(ii) Declarative. [2]

16. A name is passed as a parameter to a function. The function uses a loop structure to search for the name in an array. It returns the details found to the calling program.

(a) The name to be searched can be passed either

(i) by value, or

(ii) by reference.

Using this example, explain what is meant by a parameter being passed by value and by reference. [2]

(b) Using examples from this function, explain what is meant by a

(i) local variable,

(ii) global variable. [4]

(c) Two types of translator are interpreters and compilers.

Describe the difference between an interpreter and a compiler and state why both would be used with this function. [4]

(d) Explain the purpose of a loader in the running of the final program. [2]

17. (a) Explain the differences between the lexical analysis stage and the syntax analysis stage in the compilation of a high level language program. [6]

(b) One phase of compilation is the code generation phase.

Describe the code generation phase. [3]

(c) Explain the purpose of a loader. [2]

(d) A program has been written using a top-down technique. The individual modules in the program have been fully tested and there are no errors in any of them.

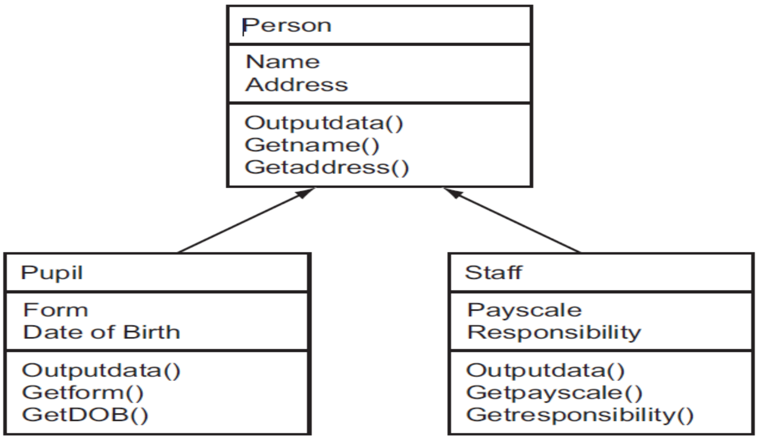
Explain why the program may fail to run or may produce incorrect results, despite the testing that has been done. [2]

(e) When a computer runs a program, the program may fail to run successfully because there are errors in the code.

Describe two types of error that may be present, giving an example of each. [6]

18. Explain why an interpreter would be preferred to a compiler as a translator when writing a high level language program. [5]

19. In a particular object oriented programming language, the following classes are defined



With reference to the diagram explain the terms:

(i) Data encapsulation

(ii) Inheritance [4]

CHAPTER 4: ALGORITHMS

- A set of instructions describing the steps followed in performing a specific task, for example, calculating change.
- They are a sequence of instructions for solving a problem.
- Algorithms can be illustrated using the following:

Descriptions, Flowcharts, Pseudocodes, Structure diagrams

a. Descriptions:

- These are general statements that are followed in order to complete a specific task.
- They are not governed by any programming language. An example is as follows:

```
Enter temperature in °C
Store the value in box C
Calculate the equivalent temperature in °F
Store the value in box F
Print the value of box C and F
End the program.
```

b. Pseudocodes:

- These are English-like statements, closer to programming language that indicates steps followed in performing a specific task.
- They are means of expressing algorithms without worrying about the syntax of the programming language.
- There are no strict rules on how pseudocode statements should be written.
- Indentations are very important in writing pseudocodes since they clearly indicate the extent of loops and conditional statements.
- They are however independent of any programming language.
- An example is as follows:

```
Enter centigrade temperature, C
If C = 0, then stop.
Set F to 32 + (9C/5)
Print C and F
End
```

Control Structures/Programming Constructs/building blocks of a structured program

- A number of **control structures** are used in designing Pseudocodes.
- These includes: simple sequence, selection and iteration.

NB: GO TO statements (also called spaghetti programming) must be avoided as the programs will be difficult to follow, difficult to debug, and difficult to maintain.

i. Simple sequence:

- This is whereby instructions are executed in the order they appear in a program without jumping any one of them up to the end of the program.
- Statements are executed one after another in the order they are.
- It is simple and avoids confusion.
- Example:

```
Enter first number, A
Enter second number, B
C = A + B
Print C
Stop
```

ii. Selection Structure:

- This allows one to choose the route to follow in order to accomplish a specific task.
- Selection is written using the **IF ....THEN...ELSE** statement or the **CASE** statement.

IF...THEN ...ELSE statement:

- A programming structure that allows the user to choose one from at least two routes of solving a problem.
- The following Pseudocodes compares two numbers entered through the keyboard and determines the bigger one.

Enter first Number, A
Enter second number, B
IF A>B THEN
        Print A is bigger
ELSE
 IF A<B THEN
        Print B is bigger
ELSE
        Print Numbers are equal
ENDIF
ENDIF
END
	Enter first Number, A
	Enter second number, B
	IF A > B THEN
	        Print A is bigger
	ENDIF
	 IF A < B THEN
	        Print B is bigger
	ENDIF
	IF A = B THEN
	        Print Numbers are equal
	ENDIF
	END
		Enter first Number, A
		Enter second number, B
		IF A>B THEN Print A is bigger
		IF A<B THEN Print B is bigger
		IF A=B THEN Print Numbers are equal
		END

A	***B***	***C***

The above 3 Pseudocodes produces the same result.

<u>Cascaded/Nested If Statements</u>

This is whereby if statements are found inside other if statements (nested Ifs) as shown below:

```
Start
    Enter "First Number", A
    Enter "Second Number", B
    Enter "Third Number", C
    If A>B Then
        If B>C Then
            Print "A is the biggest Number"
        End If
    End If
End.
```

CASE Statement: This is an alternative to the IF...THEN...ELSE statement and is shorter. For example:

```
    Enter first Number, A
    Enter second number, B
    Enter operand (+, -, * /)
    CASE operand of:
        "+": C = A + B
        "-": C = A-B
        "*": C = A*B
        "/": C = A/B
    ENDCASE
Print C
END
```

iii. Repetition/Iteration/looping:

A control structure that repeatedly executes part of a program or the whole program until a certain condition is satisfied.

Iteration is in the following forms: FOR...NEXT LOOP, REPEAT... UNTIL Loop and the WHILE...ENDWHILE Loop.

a. For...Next Loop: A looping structure that repeatedly executes the loop body for a specified number of times. The syntax of the For...Next loop is as follows:

```
FOR {variable} = {starting value} to {ending value} DO
    Statement 1
    Statement 2            loop body
    ................
NEXT {variable}
```

A group of statements between the looping structures is called the **loop body** and is the one that is repeatedly executed.

The **For...Next** loop is appropriate when the number of repetitions is known well in advance, e.g. five times. An example of a program that uses the **For...Next** loop is as follows:

```
Sum, Average = 0
FOR I = 1 to 5 DO
    Enter Number
    Sum = Sum + number
NEXT I
Average = Sum/5
Display Sum, Average
End
```

b. Repeat...Until Structure: This is a looping structure that repeatedly executes the loop body when the condition set is FALSE until it becomes TRUE. The number of repetitions may not be known in advance and the loop body is executed at least once. The syntax is as follows:

```
Repeat
        Statement 1
        Statement 2      loop body
        ................
Until {Condition}
```

For example

```
Sum, Average, Count = 0
Repeat
    Enter Number (999 to end)
    Sum = Sum + Number
    Count = count + 1
Until Number = 999
Average = Sum / count
Print Sum, count, Average
End
```

In the above program:

- **Count** records the number of times the loop body executes.
- 999 is used to stop further data entry through the keyboard and thereby ending the loop. Such a value that stops further data entry through the keyboard thereby terminating a loop is called a **Rogue value or sentinel**.

- The condition here is {**Number = 999**}. The loop exits when the number 999 is entered. If 999 is part of the number to be entered in this program, then the user has to split it into two numbers, that is 999 = 990 + 9, therefore can be entered separately as 990 and 9.
- A flag is also used to control the loop. In this case 999 is also a flag.

NB. As for the Repeat...Until loop, the condition is tested after the loop body has been run at least once, even when the condition is true from start. This is rather misleading.

c. While ... Do Structure

A looping structure in which the loop body is repeatedly executed when the condition set is TRUE until it becomes FALSE. It is used when the number of repetitions is not known in advance. The condition set is tested first before execution of the loop body. Therefore the loop body may not be executed at all if the condition set is FALSE from start. The syntax of the WHILE…ENDWHILE structure is as follows:

```
WHILE {condition}
            Statement 1
            Statement 2      loop body
            ................
ENDWHILE
```

An example of the program is as follows:

```
Sum, Count, Average = 0
WHILE Count < 6 DO
    Enter Number
    Sum = Sum + number
    Count = count + 1
ENDWHILE
Average = Sum/count
Display sum, count, average
END
```

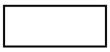
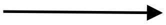
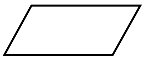
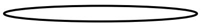
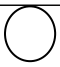
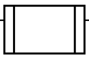
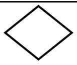
The word **WEND** can be used to replace the word **ENDWHILE** in some structures and therefore is acceptable. The word **Do, after the condition** is optional.

Differences between the Repeat...Until and the While…ENDWHILE structures

	Repeat Until Loop	**While Endwhile Loop**
1	Loop body is executed when the condition set is FALSE until it becomes TRUE	Loop body is executed when the condition set is TRUE until it becomes FALSE
2	Loop body is executed at least once	Loop body may not be executed at all
3	Condition is tested well after execution of loop body	Condition is tested before execution of loop body

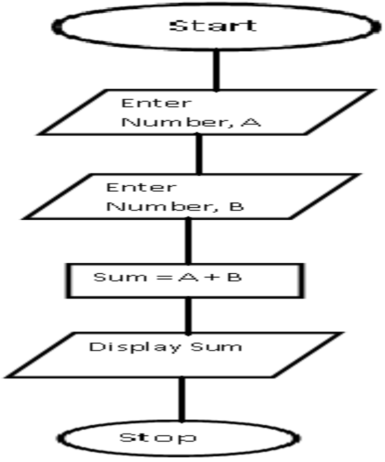
c. Flowcharts

It is a diagram used to give details on how programs and procedures are executed. Flowcharts are drawn using specific symbols, each with its own meaning, as given below:

Symbol	**Explanation**
Process Symbol 	- Indicates where some form of processing occur
Arrow 	-Shows directional flow of data (data flow symbol)
Input /output 	- Parallelogram in shape. Indicates where data is entered and output form, either screen display or printout.
Terminal 	- Oval in shape. Indicate the start and stop of a program. Therefore it is written either Start/Begin/Stop/End.
Connector 	- Circular in shape. Denotes the start and end of a subroutine. Nothing should be written inside it.
Pre-defined process 	Indicates a module/subprogram/procedure inside another program
Decision 	Represents a selection stage. Often used where a condition is, especially in repetition and selection structures.

Illustrations of flowcharts for programs

1. Using Simple Sequence Structure

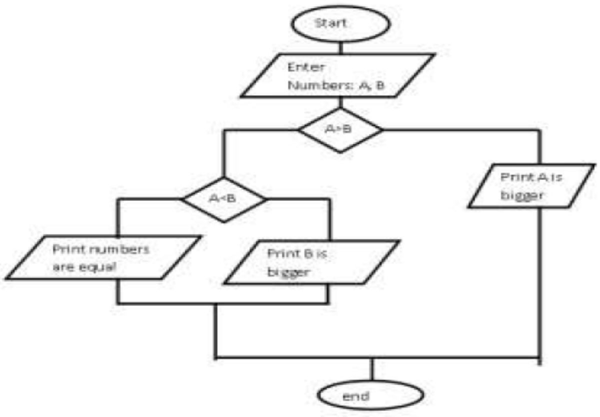


Flowchart

```
Start
Enter number, A
Enter number, B
Sum = A + B
Display Sum
Stop
```

2. Using Selection Structure

Flowchart



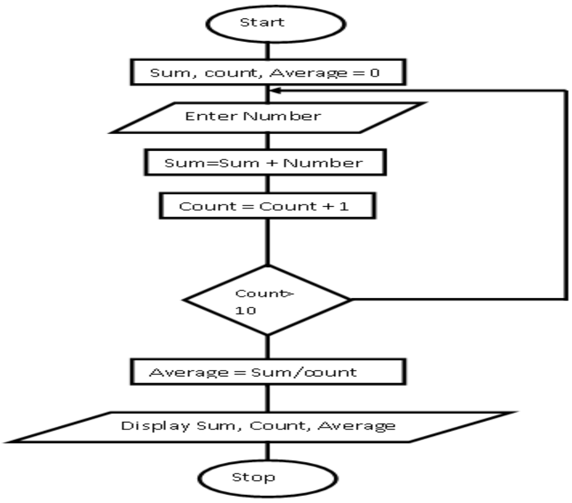
Pseudocode equivalent

```
  Enter first Number, A
Enter second number, B
IF A>B THEN
        Print A is bigger
  ELSE
IF A<B THEN
        Print B is bigger
ELSE
        Print Numbers are equal
ENDIF
ENDIF
END
```

3. Using Iteration

(a) Repeat ... Until Structure

Flowchart

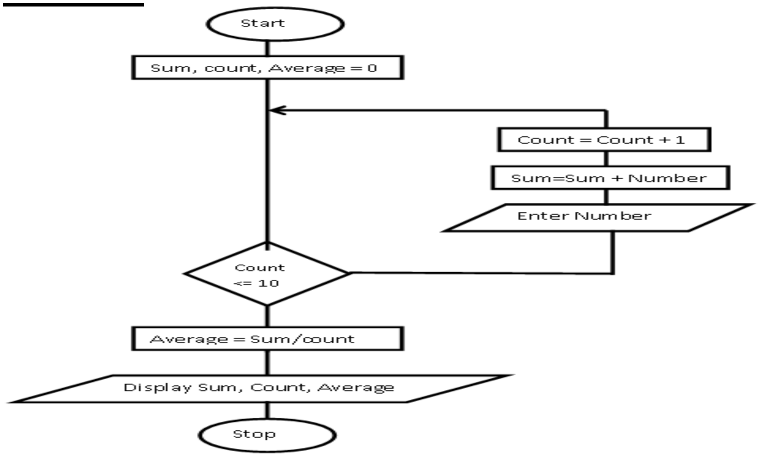


Pseudocode equivalent

```
Sum, Average, Count = 0
Repeat
Enter Number
Sum = Sum + Number
Count = count + 1
Until Count > 10
Average = Sum / count
Display Sum, count, Average
End
```

b) WHILE...WEND Structure and the FOR...TO...NEXT Loop

Flowchart

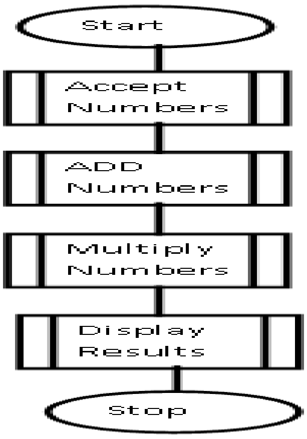


Pseudocode equivalent

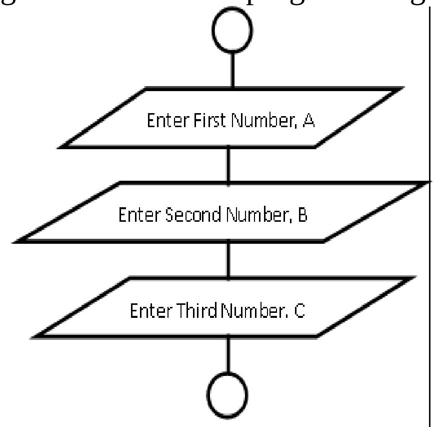
```
Sum, Average, Count = 0
WHILE Count <=10
Enter Number
Sum = Sum + Number
Count = count + 1
WEND
Average = Sum / count
Display Sum, count, Average
END
```

Use of the Pre-defined Symbol and the connector

This is used when drawing flowcharts of subprograms as given below.



a. Flowchart for whole program



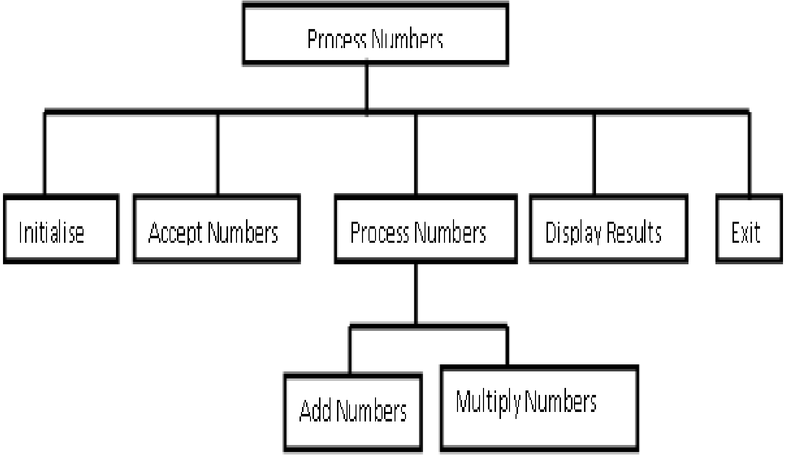
b. Flowchart for module Accept Numbers

```
Start Module Accept Numbers
Enter First Number, A
Enter Second Number, B
Enter Third Number, C
End Module
```

c. Pseudocode for module Accept Numbers

Flowchart (a) above indicates modules named Accept Numbers, Add numbers Multiply Numbers and Display Results. Flowcharts for individual modules can then be designed as given in diagram (b) above, only the first module is indicated. **Can you do the rest**?

d. Structure Diagrams/Structure Charts: These are diagrams that show relationships between different modules, thereby giving the structure of a program. They also illustrate the top-down approach to programming. It is useful as a documentation of a complex program once it is completed. It resembles a family tree as given below.



```
Start
Sum, Product = 0
Enter First Number, A
Enter Second Number, B
Sum = A + B
Product = A * B
Display Sum, Product
End
```

- The structure diagram above indicates five sub-programs of the program Process Numbers, namely Initialise, Accept Numbers, Process Numbers, Display Results and Exit.
- The module Process Numbers has its own sub-programs, which are Add Numbers and Multiply Numbers.
- Modules are appropriate for very large programs.
- If the module is repeatedly executed (loop), then an asterisk (*) must be placed at the top right corner of the module (inside).
- All the boxes at the same level indicate selection.
- Boxes below others indicate sequence.
- The program can be written as a continuous single program as indicated on the right side of the diagram.

RECURSION

A recursive function or procedure occurs when the procedure calls itself other than calling another procedure.

Recursion can be used when finding factorial of a number. For example

```
    Function Factorial (n)
        If n=1 then
            Return 1
        Else
            Return n * Factorial (n-1)
        End if
    End Function
```

A recursive structure has two important features:

- It calls itself
- It must have a terminating condition (n=1 in the above example), otherwise it will not stop calling itself (runs forever). It uses the *if condition* (not a while) to specify the terminating condition.

The following is a recursive procedure:

```
Procedure Squares (Low, High)
    If Low ≤ High Then
        Print (Low * Low)
        Squares (Low + 1, High)
    End If
End Procedure
```

- If a recursive method is called with a base case, the method returns a result. If a method is called with a more complex problem, the method divides the problem into two or more conceptual pieces: a piece that the method knows how to do and a slightly smaller version of the original problem. Because this new problem looks like the original problem, the method launches a recursive call to work on the smaller problem.
- For recursion to terminate, each time the recursion method calls itself with a slightly simpler version of the original problem, the sequence of smaller and smaller problems must converge on the base case. When the method recognizes the base case, the result is returned to the previous method call and a sequence of returns ensures all the way up the line until the original call of the method eventually returns the final result.
- Both iteration and recursion are based on a control structure: Iteration uses a repetition structure; recursion uses a selection structure.
- Both iteration and recursion involve repetition: Iteration explicitly uses a repetition structure; recursion achieves repetition through repeated method calls.
- Iteration and recursion each involve a termination test: Iteration terminates when the loop-continuation condition fails; recursion terminates when a base case is recognized.
- Iteration and recursion can occur infinitely: An infinite loop occurs with iteration if the loop-continuation test never becomes false; infinite recursion occurs if the recursion step does not reduce the problem in a manner that converges on the base case.
- Recursion repeatedly invokes the mechanism, and consequently the overhead, of method calls. This can be expensive in both processor time and memory space.

PROGRAMMING ERRORS

Programming errors are grouped into:

i. Syntax error:

- this is an error of violating the grammatical rules governing sentence construction in a certain programming language, for example, misspelled reserved words or leaving out a semi-colon at the end of each line in Pascal.
- Syntax errors are detected by the computer. A program cannot run with syntax errors.

ii. Logic error (Semantic error):

- refers to an error in the sequencing of instructions, modules and specifying wrong formulae that will produce undesirable results.
- For example, specifying a jump instruction to the wrong procedure or instructing the computer to display result before any processing has been done.
- Logic errors cannot be detected by the computer.
- The user just finds wrong and unintended results of a process.
- For example:

  NetSalary = GrossSalary + Deductions + AidsLevy

  The above formulae should have been correctly written as

  NetSalary = GrossSalary - Deductions - AidsLevy

- It is also an error generated by entering the wrong data type during program execution, for example, entering a text value where a numeric value is needed.

iii. Runtime (execution) error:

- These are errors that occur during program execution and can be generated when the computer tries to read past an end of file marker or by dividing a number by zero.

Arithmetic error: an the **arithmetic error**, occurs in an instruction which performs inappropriate arithmetic, e.g Dividing a number by zero is an arithmetic error.

Type Mismatch error

A type Mismatch error occurs in a program when a variable has been declared as one data type, but it is later assigned a value that is of an incompatible data type. The following code will produce a 'Type Mismatch' error because "Edith" is not an integer:

```
DIM MyCounter AS Integer
MyCounter = "Edith"
```

INTERPRETING AND TESTING PROGRAMS

Dry running (desk checking):

- the process of manually testing the logic of a program on paper before coding on the computer, usually using trace tables.
- Dry running is done to determine the logic of a program (to check if it gives intended results.)
- Using trace tables, one is able to trace through the program manually, writing values of variables at each stage until the end of the program or module.

Debugging:

- The process of finding and correcting errors in a program.
- Bugs are errors in a program.
- A debugger is a program used in aiding the finding and removal of errors in a program.
- A number of tolls are employed to identify and correct errors and these are:

1. Translator diagnostics.

Each of the commands that are in the original program is looked at separately by the computer translator to execute it. Each command will have a special word which says what sort of command it is. The translator looks at the special word in the command and then goes to its dictionary to look it up. The dictionary tells the translator program what the rules are for that particular special word. If the word has been typed in wrongly, the translator will not be able to find it in the dictionary and will know that something is wrong. If the word is there, but the rules governing how it should be used have not been followed properly, the translator will know that there is something wrong. Either way, the translator program knows that a mistake has been made, it knows where the mistake is and, often, it also knows what mistake has been made. A message detailing all this can be sent to the programmer to give hints as to what to do. These messages are called translator diagnostics.

2. Debugging tools.

These are part of the software which help the user to identify where the errors are. The techniques available include:

a) Cross-referencing.

This software checks the program that has been written and finds places where particular variables have been used. This lets the programmer check to make sure that the same variable has not been used twice for different things.

b) Traces.

A trace is where the program is run and the values of all the relevant variables are printed out, as are the individual instructions, as each instruction is executed. In this way, the values can be checked to see where they suddenly change or take on an unexpected value.

c) Variable dumps (check/watch).

At specified parts of the program, the values of all the variables are displayed to enable the user to compare them with the expected results.

3. Desk checking (dry run.)

The user 'works through' the program instructions manually, keeping track of the values of the variables. Most computer programs require a very large number of instructions to be carried out, so it is usual to only dry run small segments of code that the programmer suspects of harbouring an error.

Test strategies are important to establish before the start of testing to ensure that all the elements of a solution are tested, and that unnecessary duplication of tests is avoided.

Using VB 6.0, if you reach a point in your code that calls another procedure (a function, subroutine, or the script associated with an object or applet), you can enter (*step into*) the procedure or run (*step over)* it and stop at the next line. At any point, you can jump to the end (*step out*) of the current procedure and carry on with the rest of the application.

Break points can be set within program code so that the program stops temporarily to check that it is operating correctly to that point.

Step Into: Traces through each line of code and steps into procedures. This allows you to view the effect of each statement on variables.

Step Over: Executes each procedure as if it were a single statement. Use this instead of Step Into to step across procedure calls rather than into the called procedure.

Step Out: Executes all remaining code in a procedure as if it were a single statement, and exits to the next statement in the procedure that caused the procedure to be called initially.

jump to the end (step out) of the current procedure and carry on with the rest of the application

DATA TESTING

After a program has been coded, it must be tested with different data types to determine if intended results are produced. The types of test data that can be used include:

i. Extreme Data: Refers to the minimum and the maximum values in a given range. For example, a computer program requires the user to enter any number from (between) 1 to 20. 1 and 20 are extreme data and the computer must accept these. Thus extreme data is accepted by the computer.

ii. Standard (normal) Data: This refers to data that lies within (in-between) a given range. In our example above, the numbers from 2 to 19 are standard data and are accepted by the computer.

iii. Abnormal Data: This refers to data outside a given range. As to our example above, the number 0, -1, -50 and all number from 21 and above are abnormal data.

iv. Valid data: refers to data of the correct data type. Invalid data is data of the wrong data type. Thus if the user enter the value "Terrence" instead of a number, this is referred to as a wrong (invalid) data type. Only numbers are needed, not text.

PROGRAM TESTING

Can be done using the following testing methods:

Unit testing, Integration Testing, User acceptance testing, black box testing, white box testing, bottom-up testing, top-down testing, etc.

CHAPTER 5: DATA TYPE AND DATA REPRESENTATION

Data type

A data type is a method of interpreting and representing data held in a computer.

Intrinsic data types

Intrinsic data types are the data types that are defined within a particular programming language.
The **main/general** forms of data types are as follows:

Integer

- An integer is a positive or negative whole number that does not contain a fractional part.
- Integers are held in pure binary for processing and storage.
- In some programming languages integers can bell **short** and **long** integers (more bytes are used to store long integers).
- Integers are made up of digits, which are a single place that can hold numerical values between 0 and 9.
- Digits are normally combined together in groups to create larger numbers.
- For example 6357 has four digits.
- Integers are stored by the computer as binary numbers using a whole number of bytes.
- It is usual to use either 2 bytes (called short integers) or 4 bytes (called long integers), the difference being simply that long integers can store larger numbers.
- Negative integers can also be stored, these have to be treated as different types of data by the computer because the Most Significant Bit (MSB) of the data stands for something different than in an ordinary unsigned integer.

NB: The representation of integers in the computer is as follows:

- Unique bits are used to store the binary representation of the integer
- Leading zeros are used to complete the required number of bits
- Standard number of bits are used irrespective of size of integer
- Two’s complement used to represent negative integers

Real

A real is a number that contains a decimal point. Real numbers can also be referred to as single or double, depending upon the number of bytes in which they are stored.

Boolean

- A Boolean is a data-type that can store one of only two values –**True** or **False**.
- In the computer, Boolean data is stored in one byte – True being stored as 11111111 and False as 00000000.
- A simple example of its use would be in the control program for an automatic washing machine.
- One of the important pieces of information for the processor would be to know whether the door was shut.
- A Boolean variable could be set to 0 if it was open and to 1 if it was shut.
- A simple check of that value would tell the processor whether it was safe to fill the machine with water

String

- A string (or text) is a series of alphanumeric characters usually enclosed in quotation marks.
- Any type of alphabetic or numeric data can be stored as a string, e.g: “Mutare”, “07/02/1978”, “123” and “36.85” are all strings.
- Each character within a string will be stored in one byte using its ASCII code.
- However modern systems might store each character in two bytes using its Unicode.
- The maximum length of a string is limited only by the available memory.

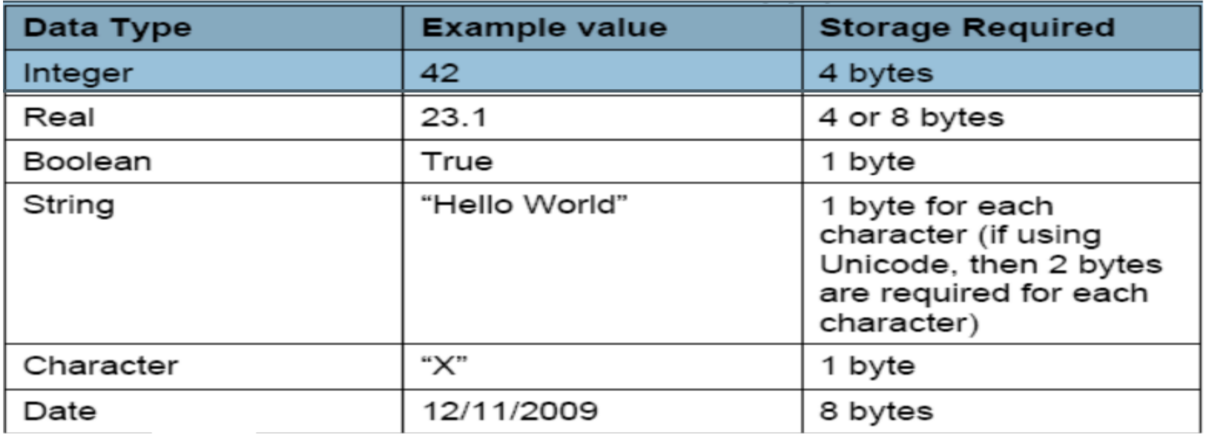
Character

A character is any single digit, letter or symbol that can be represented in a computer, for example, 2, t, G, %, &, M, space, etc. Each character is represented using binary digits, which the computer can understand; therefore take up a single unit of storage on the computer. Some programming languages refer to this as a Char. Can be used to represent coded data e.g. **M** for Male, **F** for Female.

Date

Used to represented dates, e.g date of birth, etc. can be long or short dates, e.g *dd/mm/yy, dd/mm/yyyy or dd-MonthName-yyyy*. Dates usually take 8 bytes of storage.

In general memory requirements for different data types are as follows:



Data Type	Example value	Storage Required
Integer	42	4 bytes
Real	23.1	4 or 8 bytes
Boolean	True	1 byte
String	"Hello World"	1 byte for each character (if using Unicode, then 2 bytes are required for each character)
Character	"X"	1 byte
Date	12/11/2009	8 bytes

<u>User-defined data types</u>

These are data types personally designed by programmers to suit their situation, e.g. for record designs. User-defined data types contain items from several of the different intrinsic data types.

Visual Basic uses User Defined Data types (UDTs) as a way of implementing data structures. In C/C^{++}, they are called Structures (structs) and in Pascal and COBOL they are called records.

UDTs can be declared only at the module-level (form) (you may not declare a UDT in an individual Sub or Function).
UDTs may have Public (project-level) or Private (module-level) scope. If the keyword Public or Private is omitted, the default is Public.

UDTs with Public scope may only be defined in standard modules, not forms.

The syntax for defining a UDT is:

```
 [Public/ Private] Type TypeName
        Variable1 As datatype
        ...
        Variablen As datatype
End Type
```

For example, to define a UDT for an employee record, you might code the following:

```
Public Type EmployeeRecord
        strEmpName As String
        dtmHireDate As Date
        sngHourlyRate As Single
End Type
```

To use a UDT, you must define a variable "As" the name following the keyword "Typ e" (in this case, "EmployeeRecord"). For example:

Dim udtEmpRec As EmployeeRecord
The above defines a variable called "*udtEmpRec*" which has the attributes defined by the structure "*EmployeeRecord*". Thus, it is "*udtEmpRec*" which you refer to in your procedural statements, NOT "*EmployeeRecord*". the following code places data in the individual elements of *udtEmpRec*:

```
    udtEmpRec.strEmpName = "JOE SMITH"
    udtEmpRec.dtmHireDate = #1/15/2001#
    udtEmpRec.sngHrlyRate = 25.50
```

Benefits of defined data-types

The use of data-types (Intrinsic and user-defined) within a programming language has the following benefits:

- enable the compiler to reserve the correct amount of memory for the data – e.g. 4 bytes for an integer;
- trap errors that a programmer has made and errors that a user of a program can make – a variable defined as an integer cannot be given a fractional value;
- restrict the values that can be given to the data – a Boolean cannot be given the value "maybe";
- Restrict the operations that can be performed on the data – a string cannot be divided by 10.

<u>Units Of Data Storage</u>

In general, the units of data storage are as follows:

✓	I Bit	=	1 or 0		
✓	I Nibble	=	4 Bits ($^1/_2$ a Byte)		
✓	I Byte	=	8 Bits		
✓	I Kilobyte (Kb)	=	1024 Bytes	=	2^{10} Bytes
✓	1 Megabyte (Mb)	=	1024 Kilobytes	=	2^{20} Bytes
✓	1 Gigabyte (Gb)	=	1024 Megabytes	=	2^{30} Bytes
✓	1 Terabyte (Tb)	=	1024 Gigabytes	=	2^{40} Bytes

1. <u>Bit</u>

Bit is short for **BI**nary digi**T**. It is a single digit in base 2, that is, either 1 or 0. A bit is the smallest unit of data that the computer can process. Therefore a binary number is composed of these two values only, that is 1 and 0. Bit represents two states, "ON" or "OFF", true or false, or yes or no

2. <u>Byte</u>

A byte is a group of 8 bits representing a character. A character is any digit, letter or symbol that can be represented in a computer, for example, 2, t, G, %, &, M, space, etc. Each character is represented using binary digits, which the computer can understand, therefore take up a single unit of storage on the computer. With 8 bits in a byte, you can represent 256 values ranging from 0 to 255:

0 = 00000000
1 = 00000001
2 = 00000010
...
254 = 11111110
255 = 11111111

NB: However, the byte size may differ with the architecture of the computer. Other computers use an 8-bit byte, other 32-bit byte, others 64-bit byte. Thus in general, a byte can be a unit representation of character, which could be 8, 16, 32 or in 64 bits. However, for this course, we will assume a bit as a group of 8-bits representing a character.

3. Word

A **word** is a fixed-size group of bits that can be handled as a unit by the processor.

Word size refers to the number of bits that the CPU can simultaneously process, which could be 8 bits, 16 bits, 32 bits or 64 bits. The bits are processed as a unit during input and output. A 64 bit processor can process data faster than a 32 bit processor, thus word size affects processor speed.

DATA REPRESENTATION

The form of data representation is in its character set. **All the characters that a system can recognise, which often equates to characters on the keyboard**, is called its **character set**. Character set (or data representation) can be as follows:

Each character is represented using a unique set of bits which are equivalent to 1 or 2 bytes. **Character set of a computer is represented as** binary codes, ASCII, UNICODE and EBCDic using 7/8 bits.

1. American Standard Code for Information Interchange (ASCII)

It is a set of codes that a computer understands and is represented in a single byte of 7 or 8 bits per character, which allows communication between systems. ASCII uses 7 bits which gives 128 combinations. However the extended ASCII now uses 8 bits so there are 256 different codes that can be used and hence 256 different characters. However, this is not quite true, as some of the bits can be used for parity checks.

The American Standard Code for Information Interchange (ASCII) is widely used in computers of all types.

ASCII codes are of two types –ASCII-7 and ASCII-8.

• ***ASCII-7*** is a 7-bit standard ASCII code. In ASCII-7, the first 3 bits are the zone bits and the next 4 bits are for the digits. ASCII-7 allows 27 = 128 combinations. 128 unique symbols are represented using ASCII-7. ASCII-7 has been modified by IBM to ASCII-8.

• ***ASCII-8*** is an extended version of ASCII-7. ASCII-8 is an 8-bit code having 4 bits for zone and 4 bits for the digit. ASCII-8 allows 28 = 256 combinations. ASCII-8 represents 256 unique symbols. ASCII is used widely to represent data in computers.

• The ASCII-8 code represents 256 symbols.

- Codes 0 to 31 represent control characters (non-printable), because they are used for actions like, Carriage return (CR), Bell (BEL) etc.
- Codes 48 to 57 stand for numeric 0-9.
- Codes 65 to 90 stand for uppercase letters A-Z.
- Codes 97 to 122 stand for lowercase letters a-z.
- Codes 128-255 are the extended ASCII codes.

In the ASCII character set, each binary value between 0 and 127 is given a specific character. Most computers extend the ASCII character set to use the full range of 256 characters available in a byte. The upper 128 characters handle special things like accented characters from common foreign languages.

You can see the 127 standard ASCII codes below. Computers store text documents, both on disk and in memory, using these codes. For example, if you use Notepad in Windows OS to create a text file containing the words, "Four score and seven years ago," Notepad would use 1 byte of memory per character (including 1 byte for each space character between the words -- ASCII character 32). When

Notepad stores the sentence in a file on disk, the file will also contain 1 byte per character and per space.

Try this: Open up a new file in Notepad and insert the sentence, "Four score and seven years ago" in it. Save the file to disk under the name getty.txt. Then use the explorer and look at the size of the file. You will find that the file has a size of 30 bytes on disk: 1 byte for each character. If you add another word to the end of the sentence and re-save it, the file size will jump to the appropriate number of bytes. Each character consumes a byte.

If you were to look at the file as a computer looks at it, you would find that each byte contains not a letter but a number -- the number is the ASCII code corresponding to the character (see below). So on disk, the numbers for the file look like this:

F	o	u	r		a	n	d		s	e	v	e	n
70	111	117	114	32	97	110	100	32	115	101	118	101	110

By looking in the ASCII table, you can see a one-to-one correspondence between each character and the ASCII code used. Note the use of 32 for a space -- 32 is the ASCII code for a space. We could expand these decimal numbers out to binary numbers (so 32 = 00100000) if we wanted to be technically correct -- that is how the computer really deals with things.

The first 32 values (0 through 31) are codes for things like carriage return and line feed. The space character is the 33rd value, followed by punctuation, digits, uppercase characters and lowercase characters.

ASCII codes can just be used for representing characters and not for arithmetic calculations.

ASCII codes also occupy a lot of disc storage space.

2. Binary System

Data is represented in 0s and 1s, thus in base 2. It is obtained by dividing the denary number by 2, taking the remainders only. The number of bits in the answer does not matter unless specified.

3. BCD

Each decimal digit is represented by its own 4-bit binary code as follows:

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

The number 3765 is thus coded as 0011 0111 0110 0101_2

BCD is used to represent some numbers that are **not proper numbers** (numbers that don’t behave like numbers). A barcode looks like a number, but if the barcodes are added together the result is not a barcode for any product. The arithmetic does not give a sensible answer. Values like this that look like numbers but do not behave like them are often stored in binary coded decimal (BCD). Each digit is simply changed into a four bit binary number which are then placed after one another in order.

- This has the advantage that it is easy to convert a number from BCD to decimal form and vice versa.
- There is no rounding off numbers when computing fractional numbers, thus no errors due to rounding off.
- Used in businesses where significant digit needs to be retained.

However:

- As compared to pure binary, more bits are needed to store a number, thus more memory is needed
- Calculations with such numbers are more complex than in pure binary numbers, e.g.

  Adding 1 and 19, i.e $0000\ 0001_2$
  $+\ \underline{0001\ 1001_2}$
  $\underline{0001\ 1010_2,}$ the first digit, 1 is wrong and 1010 does not exist in BCD.

The error is caused by the range of numbers used for representing data in BCD. BCD used 4 bits which is $2^4 = 16$ combinations. However the maximum range of numbers represented 9. 6 has to be added to the result if the sum of bit is greater than 9. Thus adding the result above, $0001\ 1010_2$ to 6 (0110) gives us $0010\ 0000_2$, which is 20 in BCD.

4. EBCDIC

The Extended Binary Coded Decimal Interchange Code (EBCDIC) uses 8 bits (4 bits for zone, 4 bits for digit) to represent a symbol in the data.

- EBCDIC allows 28 = 256 combinations of bits.
- 256 unique symbols are represented using EBCDIC code. It represents decimal numbers (0-9), lower case letters (a-z), uppercase letters (A-Z), Special characters, and Control characters (printable and non-printable e.g. for cursor movement, printer vertical spacing etc.).
- EBCDIC codes are used, mainly, in the mainframe computers.

5. UNICODE

Unicode is a universal character encoding standard for the representation of text which includes letters, numbers and symbols in multi-lingual environments. This is an international 16-bit data coding method which represents 65536 different characters. It is enough to represent characters in any language, even Chinese and hieroglyphics.

A problem arises when the computer retrieves a piece of data from its memory. Imagine that the data is 01000001. Is this the number 65, or is it A?
They are both stored in the same way, so how can it tell the difference?
The answer is that characters and numbers are stored in different memory locations so it knows which one it is by knowing whereabouts it was stored.

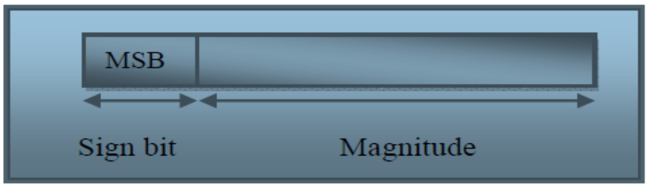
Signed and Unsigned Numbers

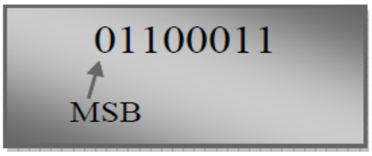
A binary number may be positive or negative. In daily life we use symbols "+" and "-" to represent positive and negative numbers, respectively. However, binary numbers use 0 (for positive) and 1 (for negative) in the computer. An *n-bit signed binary number* consists of two parts – sign bit and magnitude. The left most bit (Most Significant Bit (MSB)) is the *sign bit*. The remaining n-1 bits denote the *magnitude* of the number, giving us a sign and magnitude as given below.

In an *n-bit unsigned binary number*, the magnitude of the number n is stored in n bits. An 8-bit *unsigned number* can represent data in the range 0 to 255 (2^8= 256).

Sign and Magnitude representation

01100011 is a positive number since its sign bit is 0
11001011 is a negative number since its sign bit is 1.
An 8-bit *signed number* can represent data in the range -128 to +127 (-2^7 to $+2^7-1$).





The magnitude of a number is its natural value, regardless of the sign. Thus the magnitude of -25 and +25 is 25 (not considering the sign in this situation). In binary form, 25 = **11001**. Thus using Sign and Magnitude representation:

+25 = **0** 11001
-25 = 1 11001

What only differ is the **sign bit**, not the **magnitude**.

Representation of Negative Numbers

Negative numbers are mostly represented using the complement of number. Complement of numbers can be in 1's complement or 2's complement. The complement of a number behaves like the negative of the original number.

1's Complement

One's complement of a binary number is obtained by simply converting 1s to 0s and 0s to 1s. For example, given the following four-bit binary number 1010_2, its 1's complement becomes 0101_2. The alternating of bits only applies to negative numbers, positive numbers do not change. For example

+6 = 00000110_2
-6 = 11111001_2

-6 is the complement (negative) of +6. Just convert 1s to 0s and 0s to 1s and thus -6 in 1's complement.

Given the 1's complement we can find the magnitude of the number by taking it's 1's complement. The range of numbers that can be represented in 1's complement is found by the formula:

$-(2^{n-1}-1)$ to $+(2^{n-1}-1)$

If the binary number has 8-bits (n=8). Thus the range of numbers will be from (-127) 10000000_2 to 01111111_2 (127)

Therefore the largest number that can be represented in 8-bit 1's complement is = 127. The smallest is -127.

However 1's complement has a problem that it has two different representations (values) for zero, which are 00000000_2 and 11111111_2 both represent zero.

When **adding** binary numbers using 1's complement, the carry bit is added back to the sum in the rightmost position. There is **no overflow** as long as the magnitude of the result is not greater than 2^{n-1}-1. **We do not throw away the carry bit.**

2'S COMPLEMENT

Two's complement of number is obtained by:

a) Positive numbers remain the same
b) Negative numbers: - Change the number to its 1's complement.
- Add 1 to the result and the number will be in 2's complement.

OR

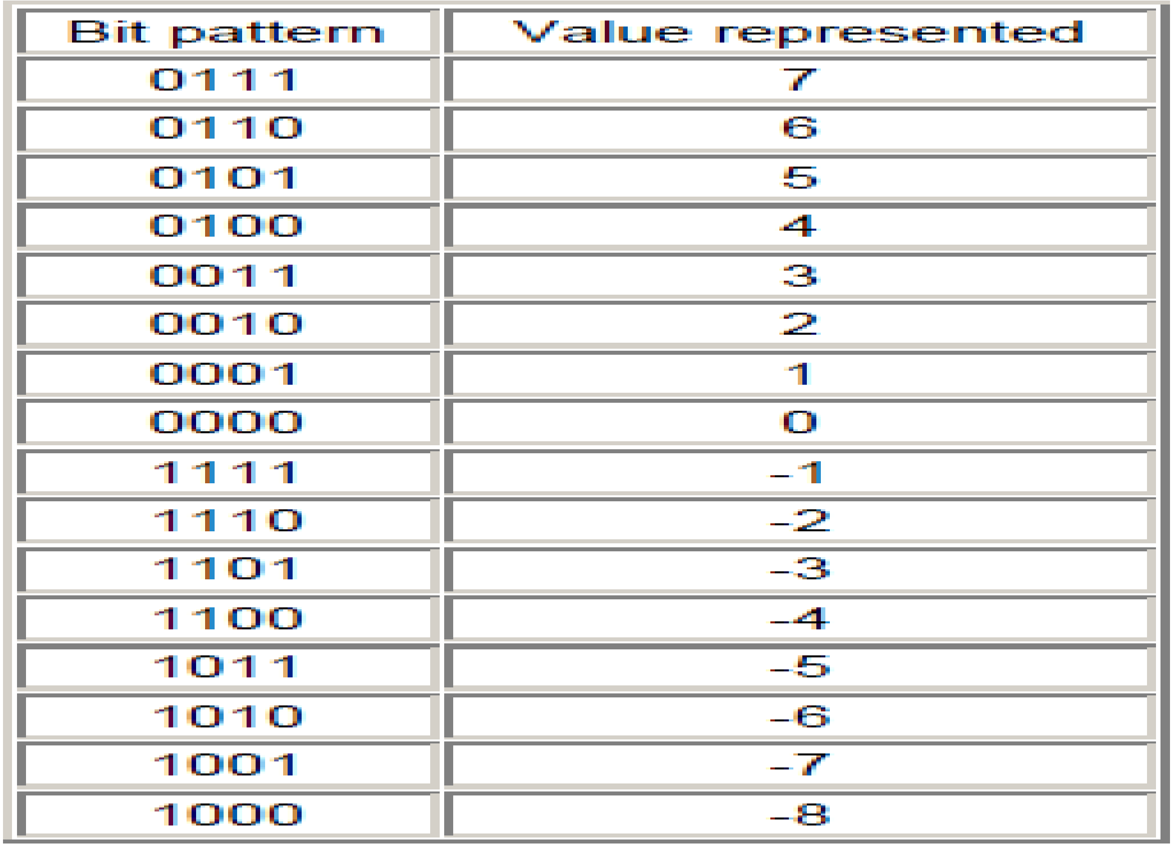
- Rewrite the bits starting from the right hand side, all 0s take as they are at their respective position and the first 1 value encountered. The rest alternate a 1 to 0 and a 0 to a 1 and your number will be in 2's complement.

For example, in 2's complement,

+6 = 00000110_2
-6 = 11111010_2

We can also find the magnitude the 2's complement number. The largest number that can be represented in 8-bit 2s complement is 01111111_2 = 127. The smallest is 100000002 = -128. The formula used for range is $-(2^{n-1})$ to $+(2^{n-1}-1)$

2’s Complement representation of 4-bit number

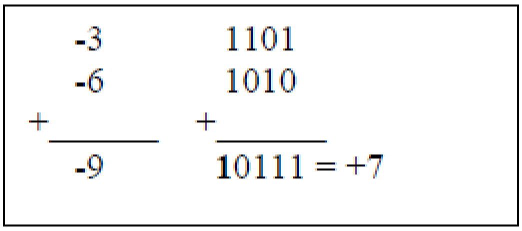


Bit pattern	Value represented
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	-1
1110	-2
1101	-3
1100	-4
1011	-5
1010	-6
1001	-7
1000	-8

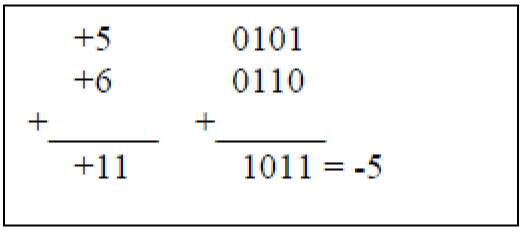
- **Range** = **Lowest Negative** =-8; **Highest positive** = +7
- Therefore range is from 1000 to 0111, it -8 to +7
- If number is added and the result is more than +7, there is an overflow.
- If the result is less than -8, it is an underflow.
- If the sign changes after addition, it’s an overflow.
- In general, overflow occurs if both numbers to be added have the same sign, otherwise no overflow occurs.

- One way to detect overflow is to check the sign bit of the sum. If the sign bit of the sum does not match the sign bit of **x** and **y**, then there's overflow. This only makes sense.
- Suppose **x** and **y** both have sign bits with value 1. That means both representations represent negative numbers. If the sum has sign bit 0, then the result of adding two negative numbers has resulted in a non-negative result, which is clearly wrong. Overflow has occurred.
- Suppose **x** and **y** both have sign bits with value 0. That means, both representations represent non-negative numbers. If the sum has sign bit 1, then the result of adding two non-negative numbers has resulted in a negative result, which is clearly wrong. Overflow has occurred.

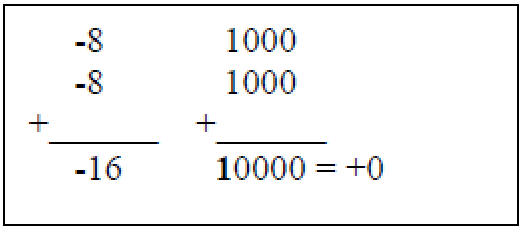
This suggests that one way to detect overflow is to look at the sign bits of the two most significant bits and compare it to the sum. Refer to diagrams below:



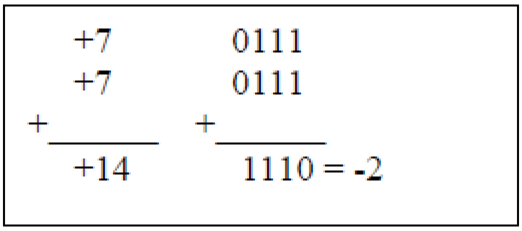
```
   -3          1101
   -6          1010
+______     +______
   -9         10111 = +7
```



```
   +5          0101
   +6          0110
+______     +______
   +11          1011 = -5
```



```
   -8          1000
   -8          1000
+______     +______
   -16        10000 = +0
```



```
   +7          0111
   +7          0111
+______     +______
   +14          1110 = -2
```

Converting From Binary Two's Complement To Denary

	1 1 1 1 1 0 1 1
Step-1 Complement the number.	0 0 0 0 0 1 0 0
Step-2 Add one add prefix a minus sign.	- 0 0 0 0 0 1 0 1
Step-3 Convert binary to decimal.	- 5

When the addition of two values results in a carry, the carry bit is ignored and is thrown away. There is no **overflow** as long as the magnitude is not greater than 2^{n-1}-1 nor less than –(2^{n-1}).
The two's-complement system has the advantage that the fundamental arithmetic operations of addition, subtraction, and multiplication are identical to those for unsigned binary numbers (as long as the inputs are represented in the same number of bits and any overflow beyond those bits is discarded from the result). This property makes the system both simpler to implement and capable of easily handling higher precision arithmetic. Also, zero has only a single representation, other than in ones'-complement where it has two values.

Binary arithmetic

The arithmetic operations - addition, subtraction, multiplication and division, performed on the binary numbers is called *binary arithmetic*. The basic arithmetic operations performed on the binary numbers are

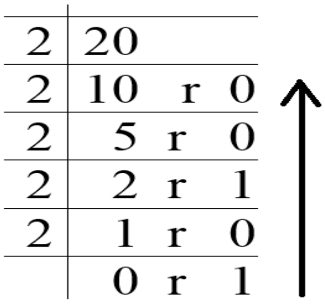
- Binary conversion
- Binary Addition, and
- Binary Subtraction,

Binary Conversion

This involves converting a number in binary from to either denary (base 10), octal (base 8) or hexadecimal(base 16)

(a) Conversion from decimal (denary) to Binary

Divide the denary number by 2, listing the remainders until the answer is 0 remainder 1.
Take the remainders only from the last one until the first.
For example:



```
2 | 20
2 | 10 r 0
2 |  5 r 0
2 |  2 r 1
2 |  1 r 0
  |  0 r 1
```

The answer is therefore 10100_2

(b) Binary to decimal conversion

Raise each bit to its binary power equivalent, from right going to the left, starting at 20

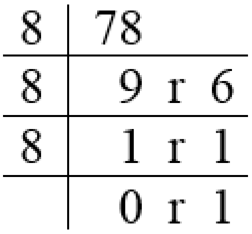
Power	2^4	2^3	2^2	2^1	2^0
Equivalent to:	16	8	4	2	1
Binary Digits	1	0	1	0	0

Add all the equivalent to values in the table, whose binary digit correspond to 1 and add them. The result is the denary equivalent.

That is 16 + 4 = 20. This is a short form of (16 x 1) + (4 x1) = 20

(c) Decimal to octal

- ✓ Octal number contains only digits from 0 to 7.
- ✓ As on binary, take the number, divide it by 8 and take the remainders only, e.g. 78 to octal will be expressed as:



```
8 | 78
8 |  9 r 6
8 |  1 r 1
  |  0 r 1
```

= $\underline{\mathbf{116_8}}$

(d) Octal to decimal

Power	8^2	8^1	8^0
Equivalent to:	64	8	1
Octal Digits	1	1	6

= (64 x 1) + (8 x 1) + (1 x 6) = **78**

(e) Decimal to hexadecimal

- ✓ Hexadecimal means base 16.
- ✓ A hexadecimal number contains numbers from 0 to 15.
- ✓ However, 10 to 15 are represented by uppercase alphabetic characters from A to F respectively.

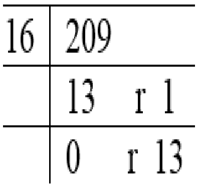
The table below illustrates this:

Decimal Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hexadecimal	0	1	2	3	4	5	6	7	8	9	**A**	**B**	**C**	**D**	**E**	**F**

Equivalent																

As on binary, take the number, divide it by 16 and take the remainders only, e.g.

209 to hexadecimal will be expressed as:



16	209
	13 r 1
	0 r 13

This gives us (13)1, but 13 is represented as D in our table above.

Thus the answer will be **<u>D1$_{16}$</u>**

(f) Hexadecimal to decimal

Power	16^1	16^0
Equivalent to:	16	1
Octal Digits	13 (D)	1

= (16 x 13) + (1 x 1) = **<u>209</u>**

(g) Binary to Octal

The relationship between binary and octal numbers is that octal (8) is 2^3. Thus binary bits are put in groups of 3, starting from the right going to the left.

If the last batch (to the left) does not have 3 bits, append 0s to its left in order to get 3 digits in each batch.

Each group is then separately converted to its decimal equivalent, which will automatically be in base 8. For example, convert 1000111010101 to octal:

i. the groupings in 3s will be as follows: 001 000 111 010 101

ii. 001, converted to decimal is = 1
000 becomes 0
111 becomes 7
010 becomes 2
101 becomes 5

The answer therefore becomes **10725$_8$**

(h) Octal to Binary

- ✓ Take each digit separately and then convert it to binary by dividing it by 2
- ✓ Put each binary result obtained into 3 bits, if not append 0s to the left
- ✓ Take the binary digits and join them into one binary number.

(i) Binary to hexadecimal

The relationship between binary and hexadecimal numbers is that hexadecimal (16) is 2^4. Thus binary bits are put in groups of 4, starting from the right going to the left.

If the last batch (to the left) does not have 4 bits, append 0s to its left in order to get 4 digits in each batch.

Each group is then separately converted to its decimal equivalent, which will automatically be in base 16.

(j) Hexadecimal to binary

Take each digit separately and then convert it to binary by dividing it by 2

Put each binary result obtained into 4 bits, if not append 0s to the left

Take the binary digits and join them into one binary number.

NB: Pupils should be able to add and subtract hexadecimal numbers, which were left out in this module. Cognisance should be taken on carry if the answer after adding exceeds 16. Bear in mind also that the decimal number 10, 11…15 and represented by letters A, B…F respectively.

Binary Addition

The table below illustrates procedure for binary addition, just like the addition of normal figures.

Input 1	Input 2	Sum	Carry
0	0	0	No carry
0	1	1	No carry
1	0	1	No carry
1	1	0	1

For 3 inputs, the table will be like:

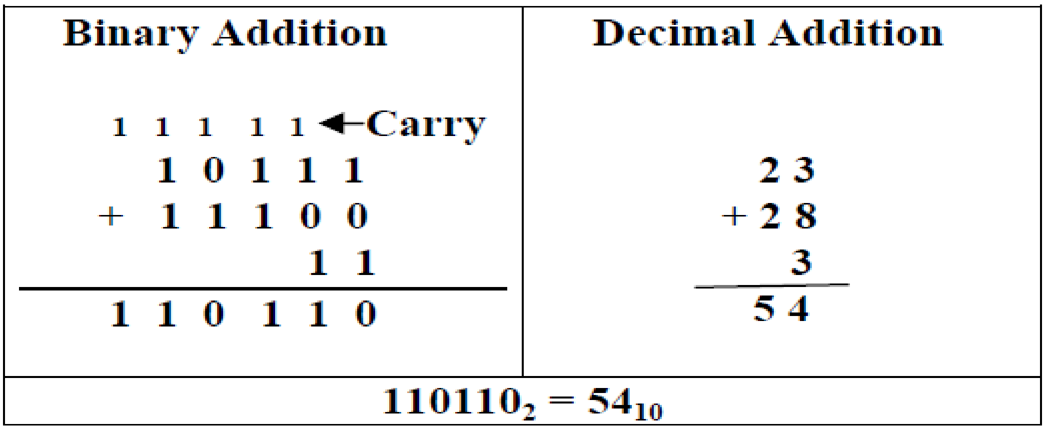
Input 1	Input 2	Input 3	Sum	Carry
1	1	1	1	1

Addition without a Carry

Binary Addition	Decimal Addition
1 0 + 0 1 Result 1 1	2 + 1 3
$11_2 = 3_{10}$	

Addition with Carry

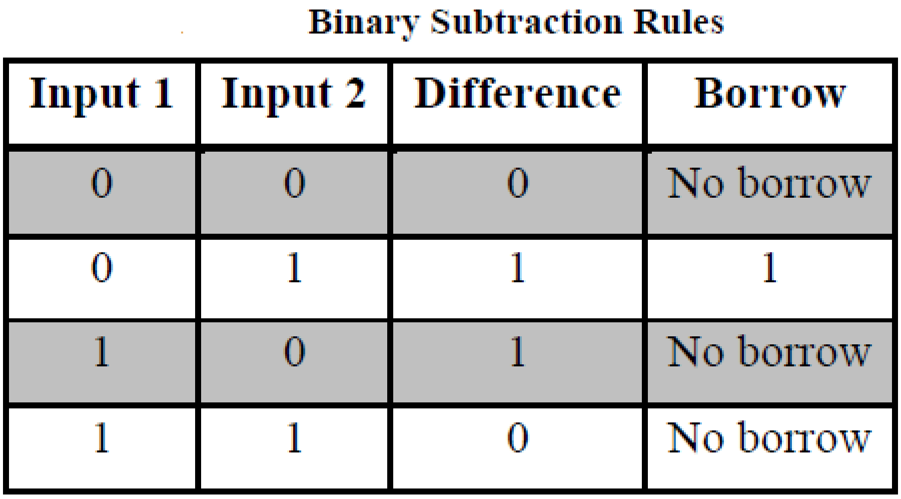
Binary Addition	Decimal Addition
1 1 ⟵ Carry 0 1 + 1 1 Result 1 0 0	1 + 3 4
$100_2 = 4_{10}$	



Binary Addition	Decimal Addition
1 1 1 1 1 ←Carry 1 0 1 1 1 + 1 1 1 0 0 1 1 1 1 0 1 1 0	2 3 + 2 8 3 5 4
$110110_2 = 54_{10}$	

Binary subtraction

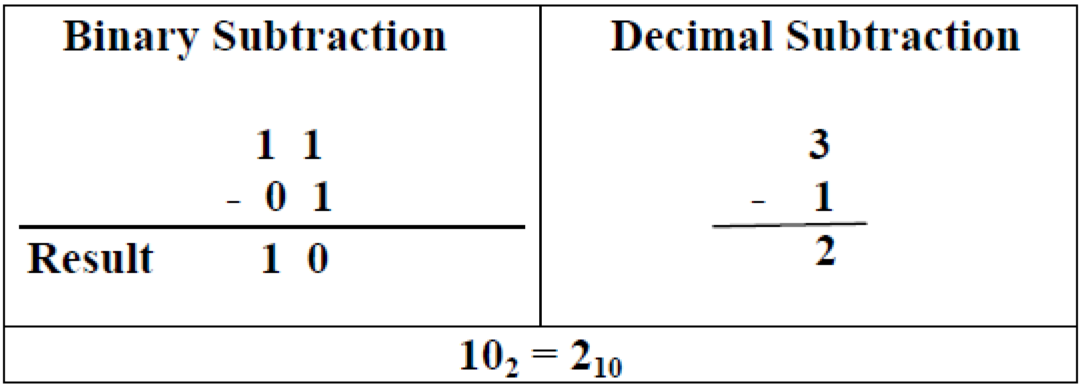
Subtraction of binary numbers follow the principles laid down on the following table:



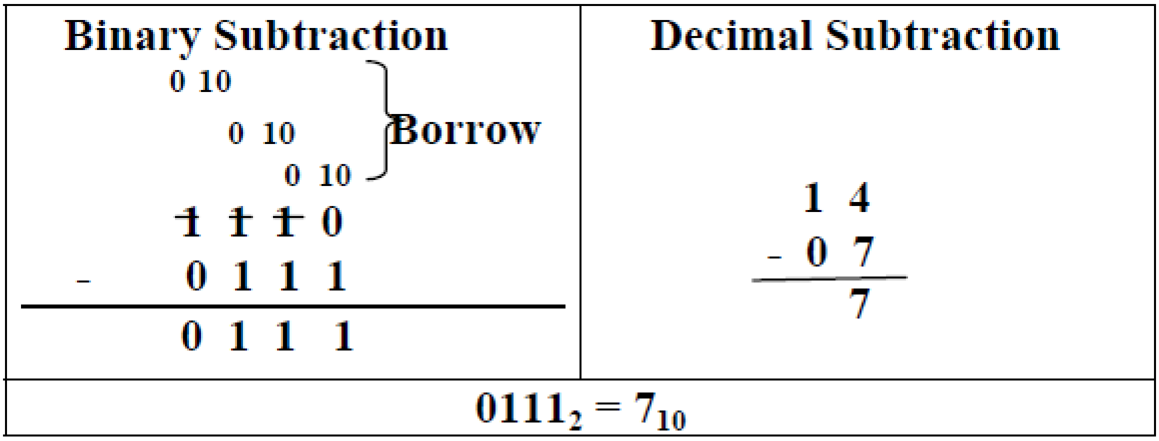
Binary Subtraction Rules

Input 1	Input 2	Difference	Borrow
0	0	0	No borrow
0	1	1	1
1	0	1	No borrow
1	1	0	No borrow

Simple examples are given below:



Binary Subtraction	Decimal Subtraction
1 1 - 0 1 Result 1 0	3 - 1 2
$10_2 = 2_{10}$	



Binary Subtraction	Decimal Subtraction
0 10 0 10 Borrow 0 10 ~~1~~ ~~1~~ ~~1~~ 0 - 0 1 1 1 0 1 1 1	1 4 - 0 7 7
$0111_2 = 7_{10}$	

Addition of signed numbers using 1's complement

When adding numbers, the carry bit is added back to the answer.

```
   1 0 0 0
 + 1 0 0 0
 1 0 0 0 0
         1
   0 0 0 1
```

Octal Numbers (Base 8)

In the octal number system there are only eight different symbols.

Decimal	Hexadecimal	Binary (4-bit)
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111

Converting Binary To Octal

Step-1 Divide the binary number into groups of three digits starting at the LSB.	111	101
Step-2 Write down the hexadecimal equivalent for each group of digits.	7	5

$111\ 101_2 = 75_8$

e.g

Express the denary number 78 as:

(i) a binary number stored in an 8 bit byte,

-Divide 78 by 2 and then take the remainders, to get 1001110. This answer has 7 bits but you are required to give your answer in 8-bit. Add a 0 to the leftmost side of the block to produce **01001110_2** which is the correct answer.

(ii) a hexadecimal number,

Divide 78 by 16 and take the remainders. This gives 4 remainder 14, and 0 remainder 4. Take the remainders only, 14 = E in base 16 and 4 remains the same. This gives the answer **$4E_{16}$**.

(iii) a number stored in binary coded decimal (BCD).

Split 78 into 2 separate digits 7 and 8. Convert each to binary separately. 7 give 111 and 8 gives us 1000. Express each binary form in 4 bits. 8 is already represented in 4 bits. Therefore add an extra bit to the left of 7 to get 0111. So the answer is **01111000**

Explain how the binary value of 78 can be used to write down the equivalent octal value with a minimum amount of calculation [3]

- The binary value of 78 is **01001110.**
- Put this binary number in groups of 3 starting from the right, and this gives us **001 001 110.**
- Each group of 3 bit can be converted separately to denary or octal form

Convert -63 and -94 into 2's complement, 8 bit, binary numbers.

- Convert the first number into its binary form by dividing it by 2 and take the remainders.
- If answer does not have 8 bits, add 0s to the left until the bits add up to 8.
- Convert it to one's complement by converting 1s to 0s and vice versa.
- Add 1 to the result and the number will be in 2's complement.
- Perform the above stages for the second number.

The results will be as follows:

- 63 in to binary form give 111111, which has 6 bits instead of 8. A leading 0 is added to the left to make them 8, thus giving us: 00111111. Change 0s to 1s and 1s to 0s, which gives us 11000000. Add 1 to the number and will give us **11000001**. Which is now in 2's complement of 64 which is -63.

- 94 will give us the answer **10100010**.

Add -63 and -94 into 2's complement

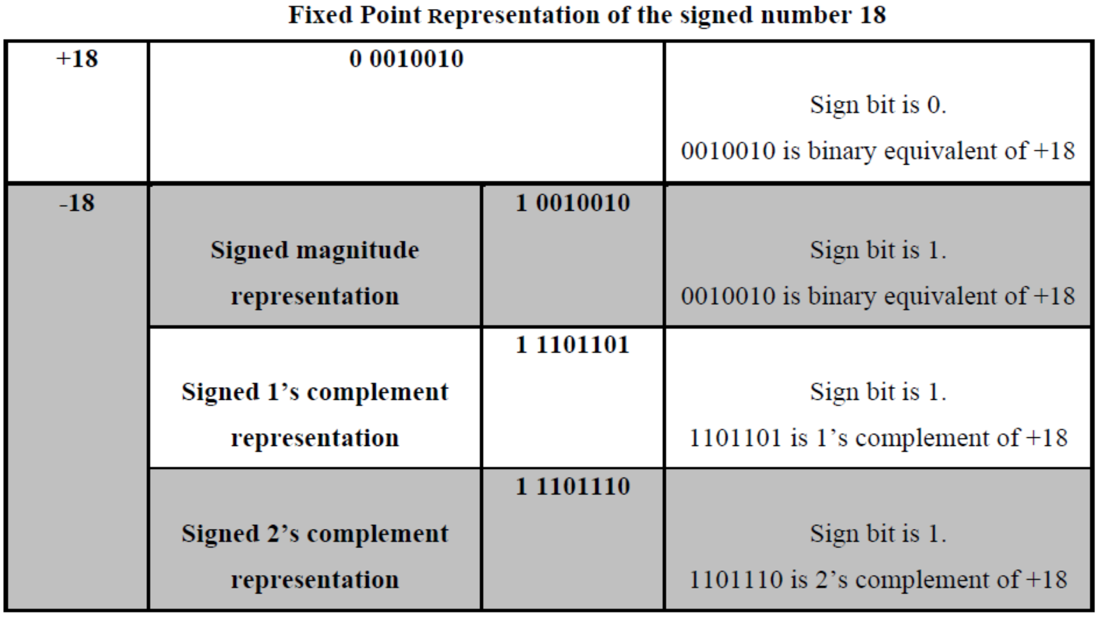
```
     11000001
     10100010
1    01100011
```

Throw carry the carry bit. To get the final answer as 01100011. The extra 1 indicates overflow.

The result above indicates overflow as the result of adding two negative numbers cannot give a positive answer. There was overflow from positive bits into negative bits. The processor will produce an error because carry in to MSB is different from carry out.

Fixed point binary number system

The *fixed point number representation* assumes that the binary point is fixed at one position. The binary point is not actually present in the register, but its presence is assumed based on whether the number which is stored is a fraction or an integer. Thus fixed point numbers can either be fractional or integer.



Fixed Point Representation of the signed number 18

+18	**0 0010010**		Sign bit is 0. 0010010 is binary equivalent of +18
-18	**Signed magnitude representation**	**1 0010010**	Sign bit is 1. 0010010 is binary equivalent of +18
	Signed 1's complement representation	**1 1101101**	Sign bit is 1. 1101101 is 1's complement of +18
	Signed 2's complement representation	**1 1101110**	Sign bit is 1. 1101110 is 2's complement of +18

Fixed Point integer representation

The decimal point in an integer is implied and does not change its position. The register used to store an integer value will be as follows:

*	*	*	*	*	*	*	*

There is no memory space for the decimal point. However computers represent a finite number of digits. This limitation allows us to evaluate the maximum and minimum possible numbers that can be represented. These include:

- **Maximum Positive Number:-**

  Positive numbers start with a 0, thus 01111111 = +127

- **Minimum Positive Number:-**

  This evaluates to 00000001 = +1

- **Smallest Magnitude Negative Number:-**

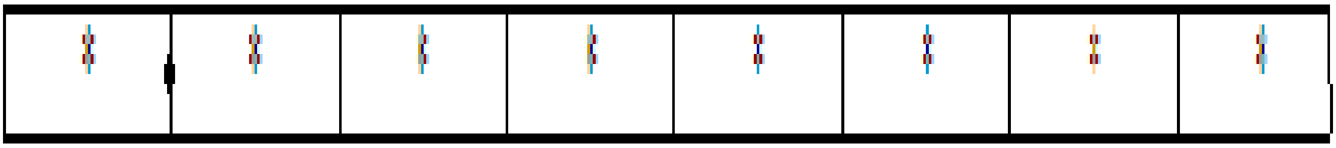
  Negative numbers start with a 1. This gives us 11111111 = -1

- **Largest Magnitude Negative Number:-**

  This is 10000000 = -128

The overall range of numbers here is -2^{n-1} to 2^{n-1}-1

Positive Fixed Point fractional Binary Numbers



- **Maximum Positive Number:-**

  This becomes 0.1111111 = 1- $^{1}/_{128}$ = 0.9921875

- **Minimum Positive Number:-**

  This evaluates to 0.0000001 = $1/2^7$=0.0078125

- **Smallest Magnitude Negative Number:-**

  Negative numbers start with a 1. This gives us 1.1111111 = $-1/2^7$=0.0078125

- **Largest Magnitude Negative Number:-**

  This is 1.0000000 = -1

The decimal point is fixed at one position and therefore does not move. In binary we can have functional column headings.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0		2^{-1}	2^{-2}
128	64	32	16	8	4	2	1		0.5	0.25
1	0	1	0	0	1	0	0	.	1	1

=164.75

Binary Fraction	**Fraction**	**Decimal**
0.1	1/2	0.5
0.01	1/4	0.25
0.001	1/8	0.125
0.0001	1/16	0.0625

4.5.2 Using Two's Complement With Fixed Point Binary Numbers

For negative numbers we use two's complement representation on the entire bit pattern.

Example

Represent -5.25_{10} in 8-bit binary with the binary point after the fourth digit.

Step-1 Calculate the positive equivalent number in binary.				5	.	2	5		
	0	1	0	1	.	0	1	0	0
Step-2 Change 0s to 1s and 1s to 0s (Complement).	1	0	1	0	.	1	0	1	1
Step-3 Add 1 to the result.	1	0	1	0	.	1	1	0	0

<u>Advantages and Disadvantages Of The Fixed Point Binary System</u>

- Preserves accuracy as required and is Easy to convert.
- However it has a limited degree of accuracy.

<u>FLOATING POINT ARITHMETIC</u>

A fixed point notation (e.g. two's complement) allows a range of positive and negative integers centred around 0 to be represented

● By assuming a fixed binary or radix point, this format would allow numbers with a fractional component to be represented

● But – this approach has limitations – **very large numbers or very small fractions cannot be represented**

● Floating point representation allows us to represent very large numbers and very small fractions

The *floating point number representation* uses two registers. The first register stores the number without the binary point. The second register stores a number that indicates the position of the binary point in the first register.

In decimal notation the number 23.456 can be written as 0.23456 x 10^2. This means that we need only store, in decimal notation, the numbers 0.23456 and 2. The number 0.23456 is called the *mantissa* and the number 2 is called the *exponent*. This is what happens in binary.

Similarly, in decimal, 0.0000246 can be written 0.246 x 10^{-4}. Now the mantissa is 0.246 and the exponent is –4.

The floating point representation of a number has two parts: **mantissa** and **exponent**. The mantissa is a signed **fixed point number**. The exponent shows the **position of the binary point** in the mantissa. For example, the binary number +11001.11 with an 8-bit mantissa and 6-bit exponent is represented as follows -

• Mantissa is 01100111. The left most 0 indicates that the number is positive.

• Exponent is 000101. This is the binary equivalent of decimal number +5.

• The floating point number is Mantissa x 2^{exponent} , i.e. + (.1100111) x 2^{+5}.

<u>**Example**</u>: consider the binary number 10111. This could be represented by 0.10111 x 2^5 or 0.10111 x 2^{101}. Here 0.10111 is the mantissa and 101 is the exponent.

Thus, in binary, 0.00010101 can be written as 0.10101 x 2^{-11} and 0.10101 is the mantissa and –11 is the exponent.

It is now clear that we need to be able to store two numbers, the mantissa and the exponent. This form of representation is called *floating point form*. Numbers that involve a fractional part, like 2.467_{10} and 101.0101_2 are called *real numbers*.

Give the denary number which would have 01000000 00000000 as its binary, floating point representation in this computer

The answer is 0.5 or ½ because it will be 0.1×2^0

It is not possible to represent zero as a normalised floating point number because a normalised value must have the first two bits of the mantissa different. Therefore one must be a 1- which must represent either -1 or + ½, but not zero.

Weaknesses of Floating Point representation

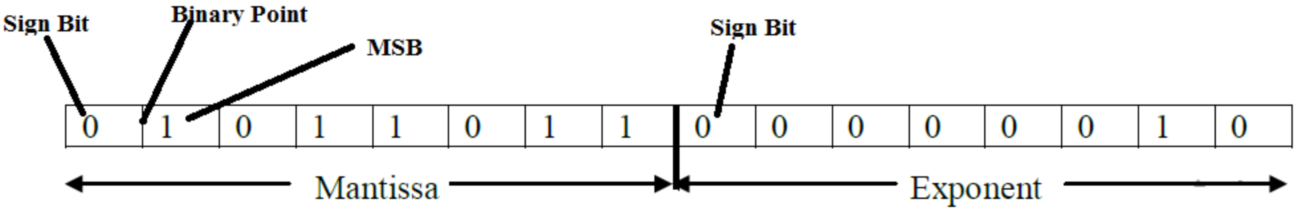
- One number can have different representations, for example, 2 (010.0000000) can be represented as
  0.100000000×2^2
  0.010000000×2^3
  0.001000000×2^4
- Causes errors as some less significant bits are lost, causing some unnecessary errors

<u>Normalisation</u>

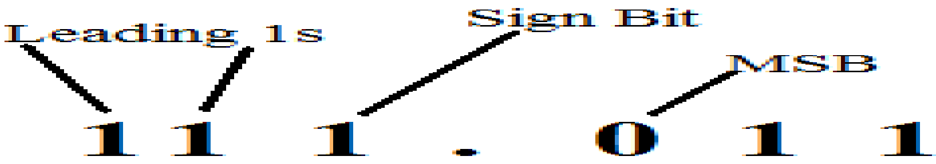
This is done to simplify operations and expressing numbers in standard form.

Normalisation Principles:

Let us look at the following diagram



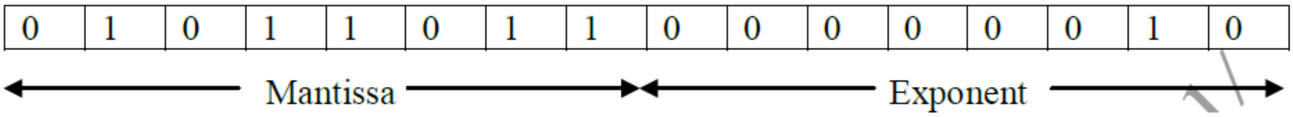
- A number is expressed in two main parts, which are:
  - The **mantissa (or fractional)** part: this is always a fraction. The binary point is always between the Sign bit and the MSB.
  - The **Exponent (or characteristic):** it is always an integer (whole number) and can be positive of negative depending on its sign bit. The left most bit of the exponent is a sign bit.
- The first two digits of the mantissa **must be different** in a normalised number. Thus:
  - If the mantissa is Positive, the Sign Bit is always 0 and the MSB is always 1.
  - If the mantissa is negative, the Sign Bit is always 1 and the MSB is always 0.
- For a negative number, there must be **NO** leading 1s to the left of the MSB, excluding the sign bit.



- For a positive number, there must be **NO** leading 0s to the left of the MSB, excluding the sign bit.

- With positive numbers, the binary point in the mantissa was always placed immediately before the first non-zero digit because it allows us to use the maximum number of digits.

Suppose we use 8 bits to hold the mantissa and 8 bits to hold the exponent. The binary number 10.11011 becomes 0.1011011 x 2^{10} and can be held as

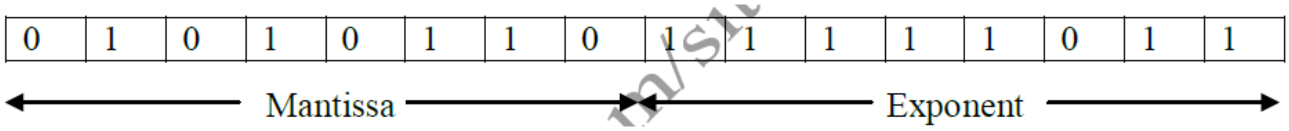


0	1	0	1	1	0	1	1	0	0	0	0	0	0	1	0

Mantissa
Exponent

The first digit of the mantissa is zero and the second is one. The mantissa is said to be normalised if the first two digits are different. Thus, for a positive number, the first digit is always zero and the second is always one. The exponent is always an integer and is held in two's complement form.

Now consider the binary number 0.00000101011 which is 0.101011 x 2^{-101}. Thus the mantissa is 0.101011 and the exponent is –101. Again, using 8 bits for the mantissa and 8 bits for the exponent, we have

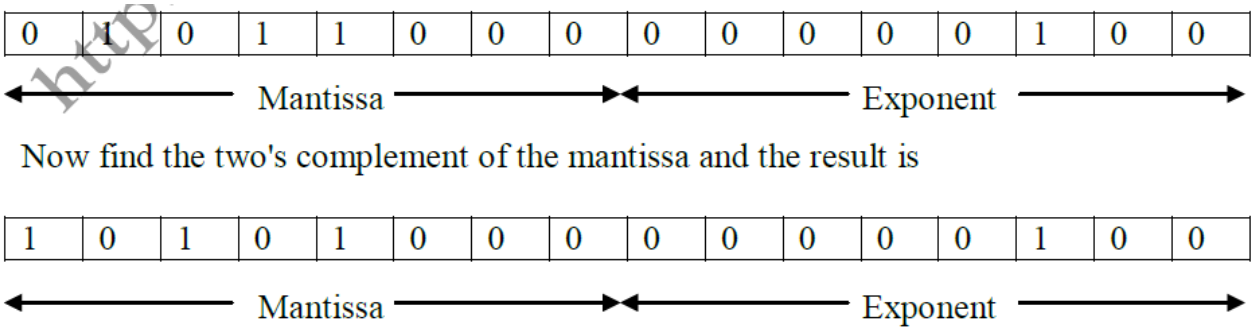


0	1	0	1	0	1	1	0	1	1	1	1	1	0	1	1

Mantissa
Exponent

Because the two's complement of –101, using 8 bits, is 11111011.
The reason for normalising the mantissa is in order to hold numbers to as high a degree of accuracy as possible.

Care needs to be taken when normalising negative numbers. The easiest way to normalise negative numbers is to first normalise the positive version of the number. Consider the binary number –1011. The positive version is 1011 = 0.1011 x 2^{100} and can be represented by



0	1	0	1	1	0	0	0	0	0	0	0	0	1	0	0

Mantissa
Exponent

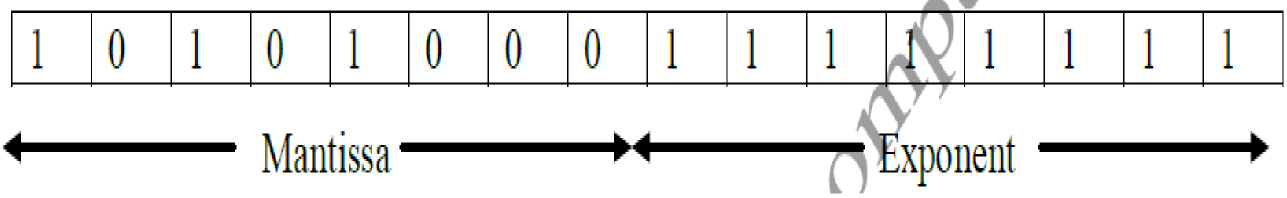
Now find the two's complement of the mantissa and the result is

1	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0

Mantissa
Exponent

Notice that the first two digits are different.

As another example, change the decimal fraction –11/32 into a normalised floating point binary number.
Leave the negative sign first and solve it for just $^{11}/_{32}$, $^{11}/_{32}$ = $^{1}/_{4}$ + $^{1}/_{16}$ + $^{1}/_{32}$ = 0.01 + 0.0001 + 0.00001 = 0.01011; converted to binary equivalent.
Now we have 8 bits mantissa and 8 bits exponent, 00101100 00000000,
It is not normalized so normalize it by removing a 0 from location worth $^{1}/_{2}$ in mantissa and subtracting that 1 location from exponent 0 reveals -1. That is, 01011000 11111111; this is Floating Point equivalent of $^{11}/_{32}$.
For $^{-11}/_{32}$ keep the exponent same and in mantissa start from right side. Keep all digits same until first 1 and toggle rest.



1	0	1	0	1	0	0	0	1	1	1	1	1	1	1	1

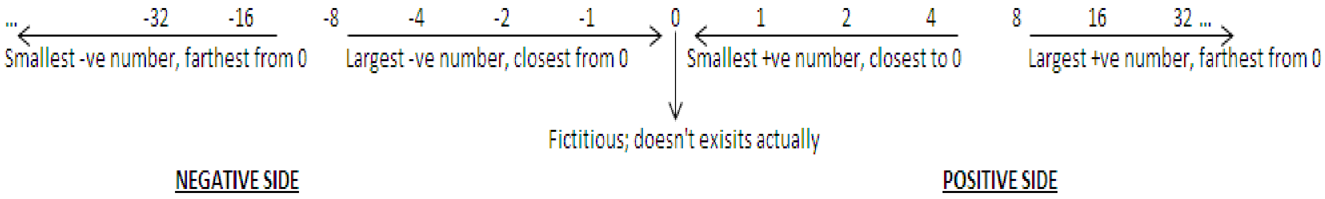
Benefits of Normalisation

- Ensures that a single representation of a number is maintained (standardisation).
- Ensures maximum possible accuracy with a given number of bit is maintained.
- Can be used to detect error conditions such as underflow and overflow
- Tires to maximise the range of numbers that can be represented in a fixed point representation (Range and accuracy is limited in fixed point representation)

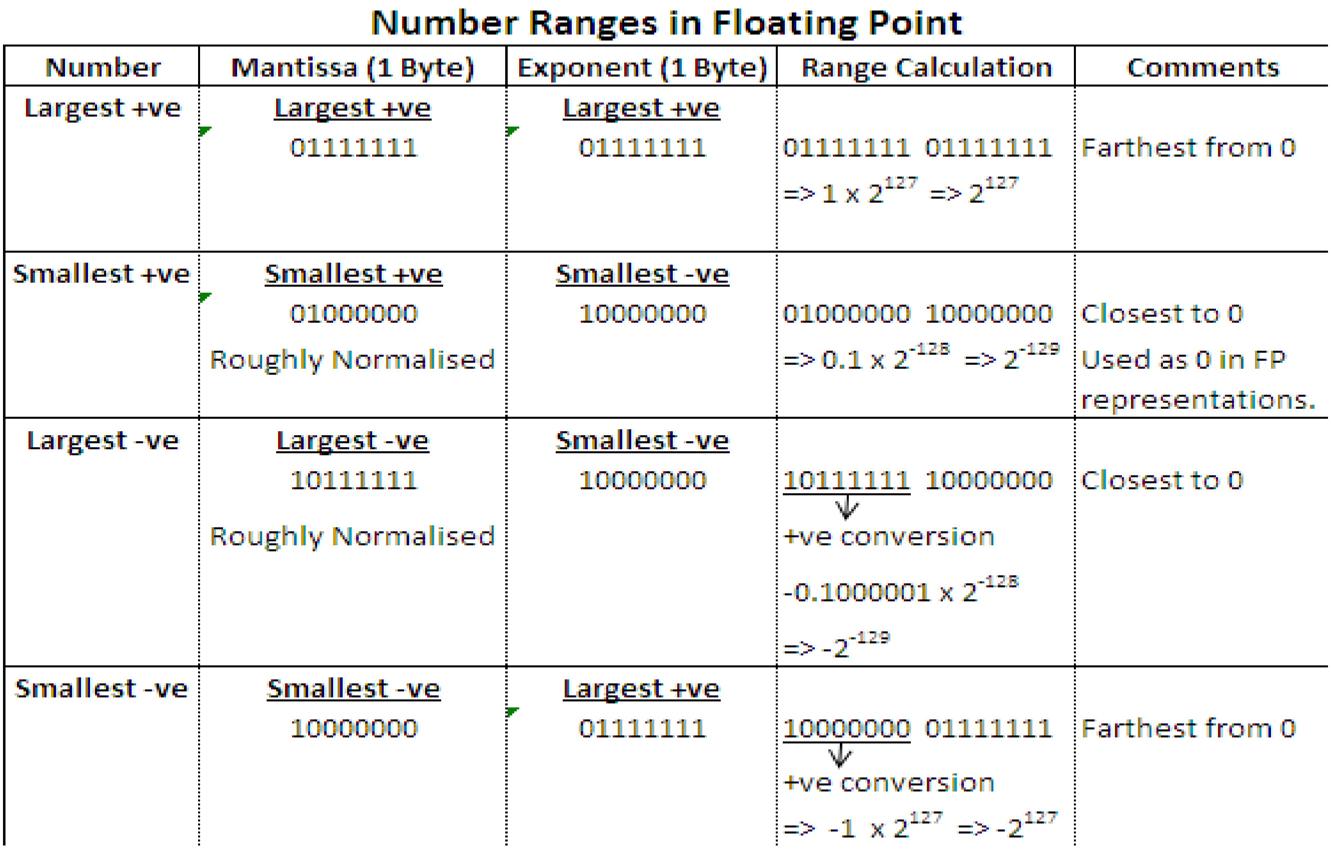
Range and Precision/ Accuracy and Range

- **The size of the exponent determines the range of numbers that can be represented**
  - ✓ The range of numbers is expanded by increasing the number of bits that are used to represent the exponent. This will however decrease precision.
  - ✓ Reducing the number of bits in the exponent will reduce the range because power of two which the mantissa is multiplying by is decreased.
  - ✓ At the same time decreasing the exponent's bits will increase accuracy because more digits are represented after the binary point.

- **The size of the significant determines the precision of the numbers that can be represented**
  - ✓ Precision can be increased by increasing the number of bits that are used to represent the significant
  - ✓ This will decrease the range.
- **The only way to increase both range and precision is to use more bits**
  - ✓ that is, the use of single-precision numbers, double-precision numbers, etc



...
-32
-16
-8
-4
-2
-1
0
1
2
4
8
16
32 ...
Smallest -ve number, farthest from 0
Largest -ve number, closest from 0
Smallest +ve number, closest to 0
Largest +ve number, farthest from 0
Fictitious; doesn't exisits actually
NEGATIVE SIDE
POSITIVE SIDE



Number Ranges in Floating Point

Number	Mantissa (1 Byte)	Exponent (1 Byte)	Range Calculation	Comments
Largest +ve	Largest +ve 01111111	Largest +ve 01111111	01111111 01111111 => 1 x 2^{127} => 2^{127}	Farthest from 0
Smallest +ve	Smallest +ve 01000000 Roughly Normalised	Smallest -ve 10000000	01000000 10000000 => 0.1 x 2^{-128} => 2^{-129}	Closest to 0 Used as 0 in FP representations.
Largest -ve	Largest -ve 10111111 Roughly Normalised	Smallest -ve 10000000	10111111 10000000 ↓ +ve conversion -0.1000001 x 2^{-128} => -2^{-129}	Closest to 0
Smallest -ve	Smallest -ve 10000000	Largest +ve 01111111	10000000 01111111 ↓ +ve conversion => -1 x 2^{127} => -2^{127}	Farthest from 0

If we use more bits for the mantissa we will have to use fewer bits for the exponent. Let us start off by using 8 bits for the mantissa and 8 bits for the exponent for explanations below:

- The **largest positive value** we can have for the **mantissa** is 0.1111111 and
- The **largest positive number** we can have for the **exponent** is 01111111.
- This means that we have 0.1111111 x $2^{1111111}$ = 0.1111111 x 2^{127}.
- This means that the largest positive number is almost 1 x 2^{127}.

Also:

- The **smallest positive mantissa** is 0.1000000 and
- the smallest exponent is 10000000.
- This represents 0.1000000 x $2^{10000000}$ = 0.1000000 x 2^{-128} which is very close to zero; in fact it is 2^{-129}.

On the other hand:

- The largest negative number (i.e. the negative number closest to zero) is 1.0111111 x $2^{10000000}$ = -0.1000001 x 2^{-128}
- We cannot use 1.1111111 for the mantissa because it is not normalised. The first two digits must be different.

Furthermore:

- The smallest negative number (i.e. the negative number furthest from zero) is 1.0000000 x $2^{01111111}$ = -1.0000000 x 2127 = -2^{127}.

Zero cannot be represented in normalised form. This is because 0.0000000 is not normalised because the first two digits are the same. A normalised value must have the first two bits of the mantissa different. Therefore one of them must be a 1 which must represent either -1 or + ½, but not zero. Usually, the computer uses the smallest positive number to represent zero.

Also, size of number mean the furthest to the left on a number line so -1 is a bigger number than -2. Whereas, if we talk about largest magnitude negative number then the -2 is greater magnitude than -1 because the integer value is greater (not considering the sign).

For a positive number n,
$2^{-129} \leq n < 2^{127}$

For a negative number n
$-2^{127} \leq n < -2^{-129}$

Range: Allowable values for a register representation of data, starting from the lowest (minimum) allowable to the highest (maximum) allowable values. It can also be defined as the difference between the lowest and the highest acceptable values.

Accuracy and Errors

Floating and fixed point numbers will be accurate to the smallest number they can represent.

Accuracy: is the degree of closeness (nearness) of measurements of a quantity to that quantity's actual (true) value.

Precision: **precision** is the degree to which further measurements or calculations show the same or similar result. More bits for precision is a measure of reliability (repeatedly give the same value). More bits for mantissa means the number will be more precise, i.e by having more significant figures

To **increase accuracy**, either you can:

- Increase the number of bits used for the mantissa
- Then reduce the number of bits for the exponent

However, the range of numbers represented is reduced because the size of the index of the power of two is reduced

Round-Off Errors

Rounding: Expressing a number to the nearest whole/decimal/binary number. For example 3.9569 rounded to 3 decimal places is 3.957. If rounded to the nearest whole number, it becomes 4.

Often we cannot represent a denary fraction exactly even if we allow many bits in memory. Therefore the number stored is "rounded off" to the closest possible binary equivalent.

In rounding, the least significant bit may be increased depending on digits removed. The result should represent the value that is nearest to the original value, e.g

100.1 = 100 (2 s. f)
10101 = 1100 (2 .f)
11011 = 11000 (2 s.f)
11.101 = 100 (2 s.f)

1100 =1100 (3 s.f)
1101 = 1110 (3 s.f)
111.1101 = 1110.11 (6 s.f)

1110.1110 = 1111.00 (6 s.f)

Truncation Errors

Truncation is shortening by cutting off some characters/ending text abruptly at a certain point; e.g. 3.9569 truncated to 3 decimal places is 3.956. If truncated to the nearest whole number, it becomes 3.

Truncation error is difference between a truncated value and the actual value. A truncated quantity is represented by a numeral with a fixed number of allowed digits, with any excess digits "chopped off"
Often, in either floating or fixed point systems, results are calculated with too many places of accuracy to be represented. We get this type of error when trailing bits are truncated to fit the result in the memory location available.

Truncation works as follows:

100.1 = 100 (2 s. f)
10101 = 10000 (2 .f)
11011 = 11000 (2 s.f)
11.101 = 11 (2 s.f)

Overflow

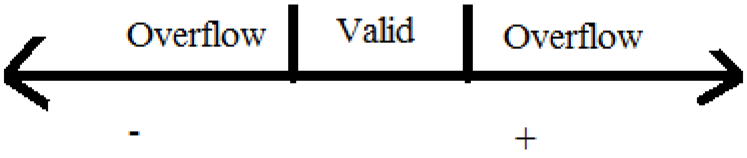
- It is a computational process produces a result so large that it cannot be represented.
- Overflow occurs when
- a number is divided by a small number or
- When two large numbers are multiplied together.

Underflow

- An underflow is produced when a result that is smaller in magnitude than the smallest number that can be represented.
- It occurs when a small number is divided by a large number or
- When small numbers are multiplied together.

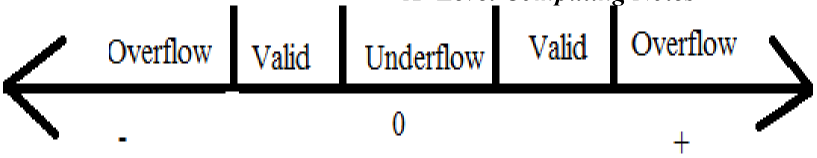
Overflow and underflow representation

(a) Integer representation



If the result is outside the possible range, then it is **overflow**.

(b) Floating Point representation



There exists the largest and smallest positive value. Around 0, there exists a range of values that cannot be represented (stored) and this is **underflow**.

NB: Overflow and underflow occur when a result of calculations falls outside the range of values permitted by the representation of the number.

Practice Questions

1. (a) Describe how characters are stored in a computer. (3)
 b) Explain what is meant by an integer data type. (2)
 c) State what is meant by Boolean data. (1)

2. a) Express the number 113 (denary) in
(i) binary
(ii) in BCD
using an appropriate number of bytes in each case. (4)
b) Using the answer obtained in part (a) show how 113 (denary) can be expressed in
(i) octal
(ii) hexadecimal. (4)

3. Explain how the denary number –27 can be represented in binary in
(i) sign and magnitude
(ii) two's complement
notation, using a single byte for each answer. (4)

4. (a) Add together the binary equivalents of 34 and 83, using single byte arithmetic, showing you working. (3)
(b). Describe a floating point representation for real numbers using two bytes. (4)

5. a) Explain how the fraction part of a real number can be normalised. (2)
b) State the benefit obtained by storing real numbers using normalised form. (1)

6. a) A floating point number is represented in a certain computer system in a single 8 bit byte. 5 bits are used for the mantissa and 3 bits for the exponent. Both are stored in two's complement form and the mantissa is normalised.
(i) State the smallest positive value,
(ii) state the most negative value
that can be stored. Give each answer as an 8 bit binary value and as a decimal equivalent. (4)
b) Explain the relationship between accuracy and range when storing floating point representations of real numbers. (4)

7. (a) Express the denary number -95 as a two's complement integer in an eight-bit byte. **[2]**
 (b) Add together the following binary numbers. Show your working.
 0 1 1 0 0 1 1 0 and 0 0 1 0 0 1 0 1 **[2]**

8. Part of the information stored in the data dictionary describes the type of data which is being stored.
A particular piece of data is 10010110.
State what the data stands for if the data dictionary describes it as:

(i) a two's complement binary number; **[1]**
(ii) a sign and magnitude binary number; **[1]**
(iii) a binary coded decimal number. **[2]**

9. Floating-point numbers in a particular computer system are stored using 12 bits. The first 6 bits are used for the storage of the mantissa and the second set of 6 bits is used to store the exponent.
(a) One way to represent 6.5 as a floating-point number in this representation is 001101000100
Explain why this representation is equivalent to 6.5 **[4]**
(b) (i) Using the representation above to help you, write the number 6.5 as a floating-point number in normalised form. **[2]**
(ii) Explain the effects of changing the representation so that 8 bits are used for the mantissa and 4 bits for the exponent. **[2]**
(c) The numbers 011011001101 and 101100001110 are stored with 6 bits for the mantissa and 6 for the exponent.
Add the exponents of the two floating-point numbers together. **[2]**

10. A computer stores floating point numbers using 2 eight-bit bytes.
The first byte is used to store the mantissa and the second stores the exponent.
(a) The normalised form of the floating point representation of $9^{1}/_{2}$ is 0100110000000100.
(i) Explain this representation of 91/2. **[4]**
(ii) Give the floating point value of 221/4 using this representation. **[2]**
(b) It is decided to change the representation by using 10 bits for the mantissa and 6 bits for the exponent.
Explain the effect of this decision on the range and accuracy of the data represented. **[4]**

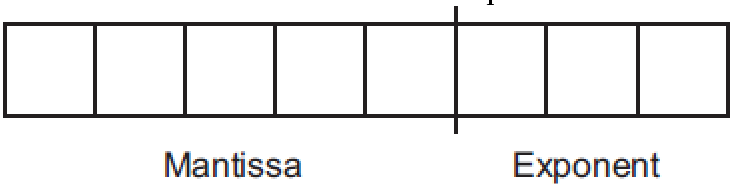
11. (a) Express the denary number 94 as:
(i) a BCD value, **[2]**
(iii) a hexadecimal value. **[2]**
(b) (i) Work out the answer to the following binary addition sum. (All values are given in two's complement form. You should show your working.)
01001101
00101011
01000101 +
__________ **[2]**
(ii) Explain why the binary result does not give the correct answer. **[1]**

12. (a) Show how the denary number –90 can be represented, using 8 bits, in:
(i) sign and magnitude,
(ii) two’s complement. **[2]**
(b) The denary number 10¾ is to be represented as a floating point binary number using 12 bits.
The first 8 bits are to be used for the mantissa and the remaining four bits are to be used for the exponent.
(i) Explain what is meant by the mantissa of a floating point number. **[2]**

(ii) Explain what is meant by the exponent of a floating point number. **[2]**
(iii) Show why 001010110101 is a floating point representation of 10¾.**[3]**
(iv) Normalise the floating point value given in (iii). **[2]**

13. (a) (i) Express the number 93 as an 8 bit binary number. **[2]**
(iii) Express the number 93 as a number in hexadecimal. **[2]**
(b) (ii) Describe the connection between binary representation and hexadecimal. [2]

14. A computer stores fractional numbers in floating point binary representation. Five bits are used for the mantissa and three bits for the exponent. All values are stored in two's complement form.



(a) By using a diagram of this representation, state the value of each of the bits. **[4]**
(b) By using 2 ½ as an example, explain how real numbers can be shown in normalised form in this representation. **[3]**
(c) State the floating point binary value of - ¾ in this representation. **[2]**

15. A computer stores numbers in floating point form, using 8 bits for the mantissa and 8 bits for the exponent. Both the mantissa and the exponent are stored in two's complement form.
(a) Explain the effect on the
• range
• accuracy
of the numbers that can be stored if the number of bits in the exponent is reduced. **[4]**
(b) Give the denary number which would have 01000000 00000000 as its binary, floating point representation in this computer. **[2]**
(c) Explain why it is not possible to represent zero as a normalised floating point number. **[2]**

16. (a) Express the decimal number 109 as
(i) a binary number stored in an 8 bit byte; **[2]**
(ii) a number in binary coded decimal (BCD); **[2]**
(iii) a hexadecimal number. **[2]**
(b) A particular computer stores numbers as 8 bit, two's complement, binary numbers.
01011101 and 11010010 are two numbers stored in the computer.
(i) Write down the decimal equivalent of 11010010. **[2]**
(ii) Add the two binary values together and comment on your answer. **[3]**

17. (a) Express the denary number 78 as
(i) a binary number stored in an 8 bit byte,
(ii) a hexadecimal number,
(iii) a number stored in binary coded decimal (BCD). **[6]**
(c) (i) Convert -63 and -94 into 2's complement, 8 bit, binary numbers. **[2]**
(ii) Add the binary values obtained in (i) together. **[2]**
(iii) Comment on the result that you obtained in (ii). **[2]**

18. (a) Express the denary value 109 as
(i) a binary number using an 8-bit byte;
(iii) a hexadecimal number. **[4]**
(b) Numbers are held in floating point form with one byte for the mantissa (fraction) and one byte for the exponent (characteristic). All values are held in two's complement form and the mantissa is normalised.
Using this format, write down the binary floating point values and the denary values of
(i) the largest magnitude, positive number;
(ii) the smallest magnitude, positive number;
(ii) the largest magnitude, negative number;
(iv) the smallest magnitude, negative number.
(The denary values may be left as a product of a power of 2). **[8]**
(c) Explain how accuracy can be improved in a floating point representation and state an effect it can have on the number represented. **[3]**

19. (a) Represent
(i) +102,
(ii) +117
as 8-bit numbers in two’s complement form **[2]**
(b) (i) Add the answers in part (a) together to give a binary result. **[2]**
(ii) Turn your binary answer into an equivalent denary result. **[2]**
(iii) Explain the validity, or otherwise, of your result. **[2]**

CHAPTER 6: COMPUTER ARCHITECTURE

Architecture refers to the structure of the processor and how computer components are related to each other.

The processor

- Can be referred to generally as Central Processing Unit (CPU) which is responsible for fetching, decoding and executing of all computer instructions.
- It is commonly called the *brain* of the computer
- Computers cannot work without the processor

The processor has the following functions:

- It controls the transmission of data from input device to memory
- It processes the data and instructions held in main memory
- It controls the transmission of information from main memory to output device.
- Controls the sequence of instructions,
- Give commands to all parts of the computer,
- Fetches the next instruction to be executed
- Decodes instructions
- Executes decoded instructions

Co-Processor

This is an additional processor used for a specific task and improves processing speed by executing jobs concurrently, e. g. maths co-processor

Maths Core-Processor: an additional processor which works alongside the main processor, capable of processing large representations using large size registers, particularly used for floating point calculations

Processor Performance

The traditional processor's performance is affected by these four main components:

(a) Clock Speed

- The processor contains a timing device known as the clock. It determines the timing of all operations.
- This sends out signals at a given interval, and all processes within the computer will start with one of these pulses.
- A process may take any amount of time to complete, but it will only start on a pulse.
- It therefore makes sense that a processor with a faster clock speed will perform faster, since more pulses will be sent out in the same time frame.
- A faster clock increases the speed of the processor and/or memory but not the peripherals
- The clock speed is generally quoted in factors of Hertz, with modern processors typically Gigahertz.

(b) Word Size

- The word size of a computer is the number of bits it can process at a time.
- Increasing the word length of the registers benefits programs with many numerical calculations
- Word length is determined by size of registers
- Bits are grouped into words, the length of a word varies but it is typically, 8, 16, 32, 64 or 128 bits.
- Obviously if the processor is able to deal with more bits at a time then it is going to perform better.
- Typically home computers are 32-bit, but 64-bit technology is becoming widespread too.

(c) Bus Width

- The addresses of data, and the data itself, is transmitted along buses inside the computer.
- The width of the bus is the number of bits it can store / carry.
- A wider data bus will allow more data to be sent at a time, and therefore the processor will perform more efficiently.
- A wider address bus will increase the number of memory addresses a computer may use.
- For example an 8 bit bus will only allow a value between 0 and 255 to be transmitted at a time.

(d) Architecture

- The architecture of a processor will affect its performance,
- A better designed processor will perform better than a different processor.

The main components of the processor are:

- ALU (Arithmetic Logic Unit)
- Control Unit (CU)
- Registers
- System Clock

1. Arithmetic and Logic Unit

Responsible for carrying out operations on data, like calculations. It consists of two parts:

(a) Arithmetic Unit
(b) Logic Unit

(a) Arithmetic Unit

Responsible for basic arithmetic functions such as: Addition, Subtraction, Multiplication, Division, etc

(b) Logical Unit

It perform logical operations like comparing two data items to find which data item is > ,= ,< the other, etc

The ALU works together with the accumulator register, which temporarily stores data being processed and the results of processing.

The ALU performs the following:

- Carries out all arithmetic.
- Carries out logic operations.
- Acts as gateway to and from the processor

2. Control unit

It manages the execution of instructions by running the clock.

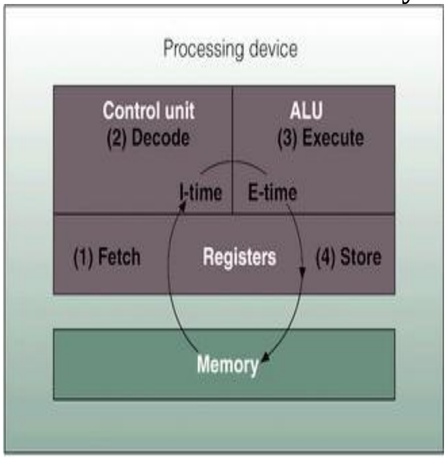
It coordinates and controls all operations of computer system.

It also called the supervisor of the computer. It performs the following:

- Fetches the next instruction to be executed
- Decodes instructions
- Manages execution of instructions
- Executes decoded instructions
- Uses control signals to manage rest of processor.

- It carries out the **Fetch-Execute Cycle as illustrated below:**

The Fetch-Decode-Execute Cycle



The Fetch-Execute Cycle

Step 1. Fetch instruction: In the instruction phase, the computer’s control unit fetches the next instruction to be executed from main memory. Microprocessor gets software instruction telling it what to do with data.

Step 2. Decode instruction: Then the instruction is decoded so that the central processor can understand what is to be done. Microprocessor determines what the instructions mean. At this stage, the computer produces signals which control other computer components like the ALU.

Step 3. Execute the instruction: In the execution phase, the ALU does what it is instructed to do, making either an arithmetic computation or a logical comparison. Microprocessor performs the instruction (cause instruction to be executed).

Step 4. Store results: Then the results are stored in the registers or in memory.

Step 3 & 4 are called the execution phase. The time it takes to complete the execution phase is called the EXECUTION TIME (E-time).

After both phases have been completed for one instruction, they are again performed for the second instruction, and so on.

(c) Registers:

- This is a **high-speed storage** area **in the CPU** used to **temporarily** hold small units of program instructions and data immediately before, during and after execution by the CPU.
- It is a small amount of storage available on the CPU whose contents can be accessed more quickly than storage available elsewhere
- Registers are special memory cells that operate at very high speed. They provide the fastest way for a CPU to access data.
- The CPU contains a number of registers and each has a predefined functions
- Most modern computer architectures operate by moving data from main memory into registers, operate on them, then move the result back into main memory
- Register size determines how much information it can store
- The size of register is in bytes: i.e., can be one, two, four or eight byte register
- The processor contains a number of special purpose registers (which have dedicated uses) and general purpose registers (which may be used for arithmetic function and are a sort of “working area”)
- The **main types** registers (special purpose registers) found in the Von Neumann Machine are as given below:
  - ✓ program counter
  - ✓ memory address register
  - ✓ memory data register/memory buffer register
  - ✓ current instruction register
  - ✓ index register

Program Counter (PC)

- Contains the address of the next instruction to be fetched/executed
- PC holds address of next instruction
- this register is automatically incremented so that it always holds the memory address of the next instruction
- Keeps check of whereabouts the next program is in the memory.
- After one instruction has been carried out, the PC will be able to tell the processor whereabouts the next instruction is.
- The PC is also called the Sequence Control Register (SCR) as it controls the sequence in which instructions are executed.
- During program execution, the PC:
  - stores the address of the next instruction to be executed
  - Its content is incremented after the address is read
  - Its content is altered to specific address if instruction is a jump instruction

Memory address register (MAR)

- Is used to hold the memory address that contains either the next piece of data or an instruction that is to be used
- It holds the address of a memory location from which data will be read from or written to. Data might be a variable as part of a program, or an instruction for the processor to execute
- Specifies the address for the next read or write
- MAR holds address of instruction/data
- This is where the address that was read from the PC is sent.
- Stored here so that the processor knows where-abouts in the memory the instruction is.

Memory Data Register (MDR) also called Memory buffer register (MBR)

- This is used to store data which has been read from or is ready to write to memory. All transfers from memory to CPU go through this buffer
- It acts like a buffer and holds anything that is copied from the memory ready for the processor to use it
- It contains data written into memory or receives data read from memory

Current instruction register (CIR)

- it holds the instruction that is to be executed
- Contains both the operator and operand of the current instruction
- For example the instruction ***MOVE 100,#13***
- "MOVE" is the operator, and "100" and "#13" are the operands.
- It stores an instruction while it is being decoded/executed/carried out
- Its contents change when an instruction from memory has been placed in MDR, and then it is copied from MDR to CIR.

The Status Register (SR).

- This contains status bits which reflect on the results of an instruction. For example, if there is an error in the operation (such as an overflow) this will be recorded in the status register. The Status Registers also store data about interrupts

Instruction Register (IR)

- Contains the instruction most recently fetched or the instruction currently being executed

Index register

- It is a register used for modifying operand addresses during program execution,
- Used in performing vector/array operations.
- Used for indirect addressing where an immediate constant (i.e. which is part of the instruction itself) is added to the contents of the index register to form the address to the actual operand or data

How and why is the index register (IR) used

- used in indexed addressing
- stores a number used to modify an address which is given in an instruction
- allows efficient access to a range of memory locations by incrementing the value in the IR e.g. used to access an array
- It stores an integer value
- The integer value is added to the base address in the instruction
- Used for the successive reading of values from memory locations e.g. in an array
- Can be incremented after use

General Purpose Registers

Accumulator

- A general purpose register used to accumulate results of processing
- it is where the results from other operations are stored temporarily before being used by other processes.
- Available to the programmer and referenced in assembly language programs
- Used for performing arithmetic functions

Flags Register: Used to record the effect of the last ALU operation

HOW REGISTERS OPERATE

Fetch phase:

- The address of next instruction is copied from the PC to the MAR.
- The instruction held at that address is copied to the MDR
- The contents of the MDR are copied to the CIR

Execute Phase:

- The instruction held in the CIR is decoded.
- The instruction is executed.

The Fetch-Decode Execute Cycle

There are three stages of the machine cycle in a Von Neumann architecture, which are: fetch, decode and execute stages.

The stages are summarised as follows:

Fetch

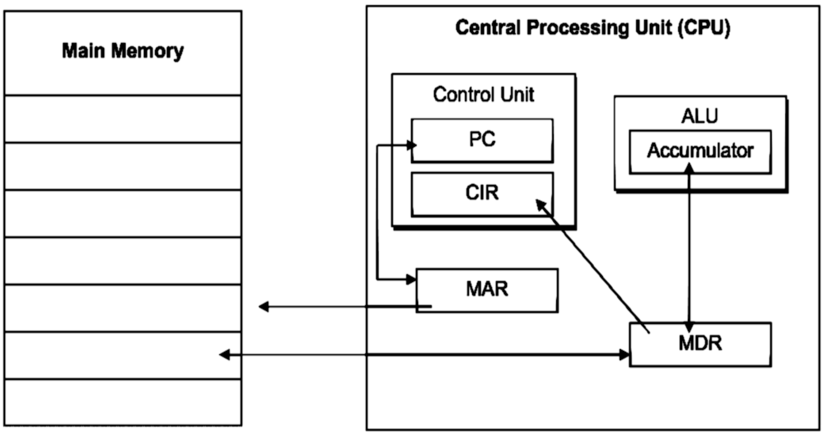
- The PC stores the address of the next instruction which needs to be carried out
  As instructions are held sequentially in the memory, the value in the PC is incremented so that it always points to the next instruction.
- When the next instruction is needed, its address is copied from the PC and placed in the MAR
- The data which is stored at the address in the MAR is then copied to the MDR
- Once it is ready to be executed, the executable part if the instruction is copied into the CIR

Decode

- The instruction in the CIR can now be split into two parts, the address and the operation
- The address part can be placed in the MDR and the data fetched and put in the MAR.

Execute

- The contents of both the memory address register and the memory data register are sent together to the central processor. The central processor contains all the parts that do the calculations, the main part being the CU (control unit) and the ALU (arithmetic logic unit), there are more parts to the central processor which have specific purposes as well.
- The ALU will keep referring back to where the data and instructions are stored, while it is executing them, the MDR acts like a buffer, storing the data until it is needed
- The CU will then follow the instructions, which will tell it where to fetch the data from, it will read the data and send the necessary signals to other parts of the computer.



The fetch-decode-execute cycle operates in the following way:

- Load the address that is in the program counter (PC) into the memory address register (MAR).
- Increment the PC by 1.
- Load the instruction that is in the memory address given by the MAR into the MDR
- Load the instruction that is now in the MDR into the current instruction register (CIR).
- Decode the instruction that is in the CIR.
- If the instruction is a jump instruction then
  - Load the address part of the instruction into the PC
  - Reset by going to step 1.
- Execute the instruction.
- Reset by going to step 1.

How a jump instruction executed

- by changing contents of PC (to address part of
- instruction)
- copy address part of instruction in CIR to PC

The first step simply places the address of the next instruction into the memory Address Register so that the control unit can fetch the instruction from the correct part of the memory. The program counter is then incremented by 1 so that it contains the address of the next instruction, assuming that the instructions are in consecutive locations.

The memory data register is used whenever anything is to go from the central processing unit to main memory, or vice versa. Thus the next instruction is copied from memory into the MDR and is then copied into the current instruction register.

Now that the instruction has been fetched the control unit can decode it and decide what has to be done. This is the execute part of the cycle. If it is an arithmetic instruction, this can be executed and the cycle restarted as the PC contains the address of the next instruction in order. However, if the instruction involves jumping to an instruction that is not the next one in order, the PC has to be loaded with the address of the instruction that is to be executed next. This address is in the address part of the current instruction, hence the address part is loaded into the PC before the cycle is reset and starts all over again.

Memory Unit

Is the computer memory that temporarily stores the operating system, application programs and data currently use.

It used to store the following:

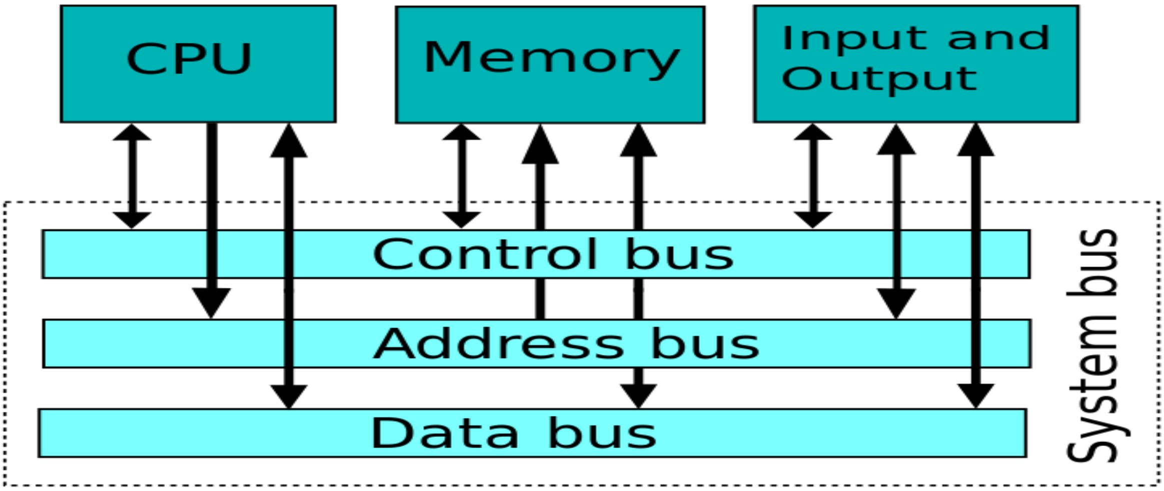
- Program instructions in current use;
- Data in current use;
- Parts of Operating System that are currently in use.

Some architectures have a Memory Unit (Main memory) which has two types: RAM and ROM.

Buses

- A bus is a pathway through which data and signals are transferred from one device to another.
- They are a set of parallel wires connecting two or more components of the computer.
- Buses can be internal or external.
- Buses can be generally referred to as **system** bus and this connect the CPU, memory and I/O devices.
- Each bus is a shared transmission medium, so that only one device can transmit along a bus at any one time.
- Multiple devices can be connected to the same bus

Diagram



- Data and control signals travel in both directions between the processor, memory and I/O controllers.
- Address, on the other hand, travel only one way along the address bus: the processor sends the address of an instruction, or of data to be stored or retrieved, to memory to an I/O controller.

The main types of buses are:

- **Data bus**:
  - Used for carrying data from memory to the processor and between I/O ports.
  - Comprises of either 8, 16, 32 or 64 separate parallel lines
  - Provide a bi-directional path for data and instruction's between computer components. This means that the CPU can read data from memory and input ports and also send data to memory and output ports.
  - The width of the bus determines the overall system performance. For example, if the data bus is 8 bits wide, and each instruction is 16 bits long, then the processor must access the main memory twice during each instruction cycle

- **Address bus**:
  - Used for transferring memory addresses from the processor when it is accessing main memory
  - They are used to access memory during the read or write process
  - The width of the address bus determines the maximum possible memory capacity of the computer.
  - This a uni-directional bus (one way). The address is send from CPU to memory and I/O ports only.

- **Control bus**:
  - The purpose of the control bus is to transmit command, timing and specific status information between system components. Timing signals indicate the validity of data and address information. Command signals specify operations to be performed. Specific status signals indicate the state of a data transfer request, or the status of request by a components to gain control of the system bus
  - This is a bi-directional bus used for carrying control signals (Signals can be transferred in both directions).
  - They carry signals to enable outputs of addressed port and memory devices
  - Control signals regulate activities on the bus.
  - Control buses transmit command, timing and status information between computer components.
  - Typical control signals are:
    - ✓ Memory Read
    - ✓ Memory Write
    - ✓ I/O Read
    - ✓ I/O Write
    - ✓ Interrupt Request
    - ✓ Interrupt Grant
    - ✓ Reset
    - ✓ Ready hold
    - ✓ etc
  - **Timing signals**: indicate validity of data and information.
  - **Command signals**: Specify operations to be performed
  - **Status signals**: Indicate state of data transfer request or status of a request.

INTERRUPTS

Interrupt: it is a signal generated by a device or software, which may cause a break in the execution of the current routine

An interrupt is a signal send to the processor by a peripheral or software for attention to be turned to that peripheral/software, thereby causing a break in the execution of a program, e.g. printer out of paper.

Control is transferred to another routine and the original routine will be resumed after the interrupt

Interrupt Service Routine (Handler): a small subprogram that is calld when an interrupt occurs and it handles the interrupt.

Interrupt priorities

Interrupts have different priorities

This is important if two interrupts are received simultaneously the processor for it to decide which one is more important to execute first.

There are four levels of priority, which are (highest priority order):

- Hardware Failure: can be caused by power failure or memory parity error.
- Program Interrupts: Arithmetic overflow, division by zero, etc
- Timer Interrupts: generated by the internal clock
- I/O Interrupts:

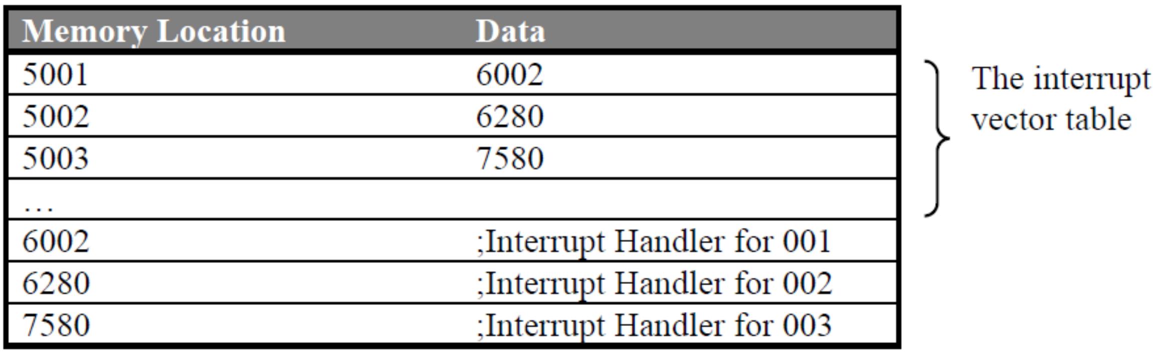
Interrupt Handling

At the end of each Fetch-Execute cycle, the contents of the interrupt registers are checked.

Should there be an interrupt; the following steps will typically be taken:

a) The current fetch-decode-execute cycle is completed
b) The operating system halts current task
c) The contents of the PC and other registers will be stored safely in a stack.
d) The highest priority interrupt is identified. Interrupts with a lower priority are disabled.
e) The source of the interrupt is identified.
f) The start address of the interrupt handler is loaded into the PC.
g) The interrupt handler is executed.
h) Interrupts are enabled again, and the cycle will restart with any further interrupts.
i) The PC and other registers are "popped" from the stack and restored.
j) The user's program resumes with the next step in its cycle.

- When dealing with an interrupt, the computer has to know which interrupt handler to call for which interrupt.
- One method of doing this is known as the vectored interrupt mechanism.
- In this approach a complete list of interrupts and the memory address of their handler is stored in a table called the interrupt vector table.
- The interrupt supplies an offset number, which identifies the interrupt uniquely.
- This offset is added to a base term, and the resultant number is the memory address of a *pointer* to the memory location of the handler routine.
- This is explained in the example below:



Memory Location	Data
5001	6002
5002	6280
5003	7580
…	
6002	;Interrupt Handler for 001
6280	;Interrupt Handler for 002
7580	;Interrupt Handler for 003

The interrupt vector table

- If the interrupt 002 is received, the base number 5000 is added to it, which allows the processor to know that the handler can be found by opening the data stored at address 5002.
- The address 5002 simply stores a pointer to another memory location, 6280, where the actual handler routine begins.
- The advantage of this approach is that each interrupt only needs to give the processor an offset number, such as 002, and the processor can determine from that the correct memory location to use. This is more efficient than the interrupt sending the full memory address itself. This approach also allows the interrupt routines to be stored anywhere in the memory, with the pointer table updated to reflect if a handler routine is moved

<u>Types of interrupts</u>

Input / output interrupt e.g. disk full, printer out of paper, etc. they are generated by the I/O devices
Interrupts generated by running process: process may need more storage or to communicate with the operator
Timer interrupts: generated by the processor clock, e.g. control being transferred to another user in a time sharing system
Program check interrupts: caused by errors like division by zero
Machine check interrupts: Caused by malfunctioning hardware**.**
Clock (happens normally in time sharing systems where the clock transfers control from one computer to another.)

Why interrupts are used in a computer system

- to obtain processor time for a higher priority task
- to avoid delays
- to avoid loss of data
- as an indicator to the processor that a device needs to be serviced
- allows computer to shut down if the power off interrupt predicts loss of power, saving data in time

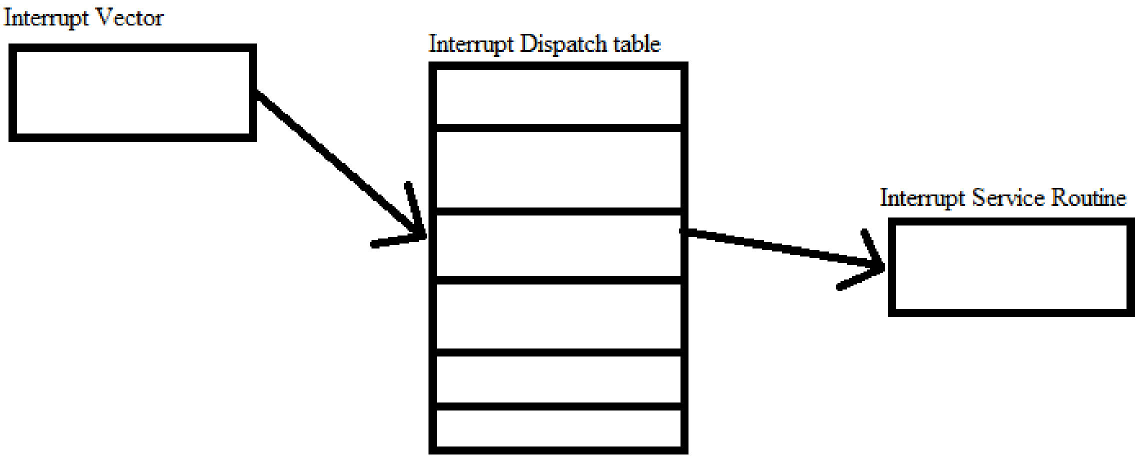
Sources of interrupts

- power failure/system failure
- peripheral e.g. printer (buffer empty)/hardware
- clock interrupt
- user interrupt e.g. new user log on request
- software

Vectored Interrupts

A specific number assigned to each interrupt is called an interrupt vector. Each interrupt is numbered. Each interrupt vector is the one used to call the interrupt handler

Address of interrupt service routines are stored in an array (known as interrupt dispatch table) and the interrupt vector is used as a subscript to this array.



Buffer

- **Buffer:** This is a temporary memory store for data awaiting processing or output, compensating speed at which devices operate, for example printer buffer.
- A buffer is a memory in the interface between two devices which temporarily store data which is being transmitted from one device to another
- A buffer is a small amount of fast memory outside the processor that allows the processor to get on with other work instead of being held up by the secondary device.
- The buffer is necessary if the two devices work at the different speed
- Buffering is appropriate where an output device processes data slower than the processor. For example, the processor sends data to the printer, which prints much slower and the printer does not need to wait for the printer to finish printing in order for it to carry out the next task.
- It therefore saves the data in a buffer where it will be retrieved by the printer.
- Buffering usually match devices that work at different speeds, e.g. processor and disk.
- Sometimes a device is already busy in executing some instructions.
- **Example**: there are three printing jobs, the printer can print only one job at a time. The OS sends the next two jobs in buffer, a process which is also known as Spooling
- Buffers are a main component of Memory
- The printer buffer is one of the most common type of buffer.

Reasons for using printer buffers:

Stores data or information being sent to the printer temporarily.
Compensates for difference in speed of CPU and printer.
Allows CPU to carry out other tasks whilst printer is printing.

Benefits of increasing size of buffer in a printer:

Reduces the number of data transfers to the printer.
Ensures a more efficient use of the CPU.
Larger files can be sent to the printer without problems

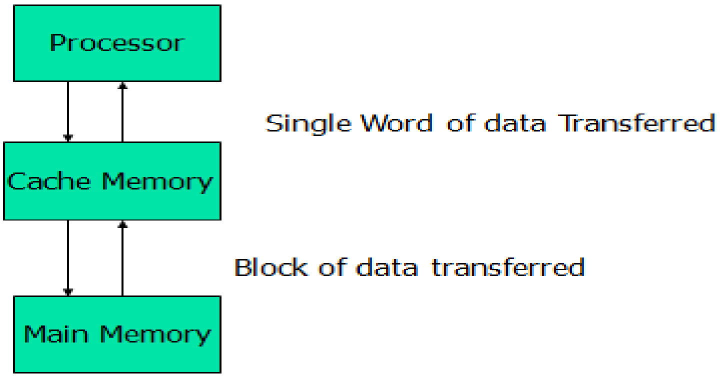
Use of buffers and interrupts in the transfer of data between primary memory and hard disk.

- Buffer is temporary storage area for data
- Data transferred from primary memory to buffer (or vice versa)
- When buffer full, processor can carry on with other tasks
- Buffer is emptied to the hard disk
- When buffer empty, interrupt sent to processor requesting more data to be sent to buffer.
- Works according to priorities

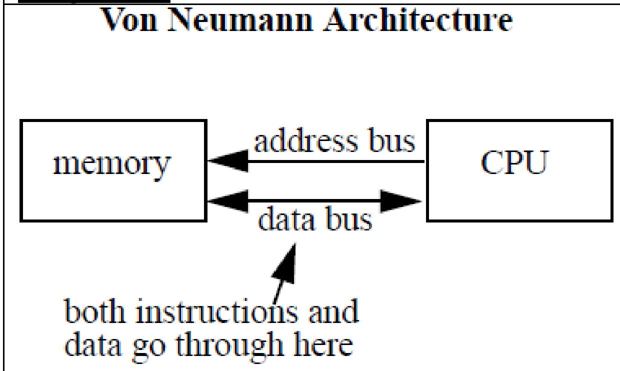
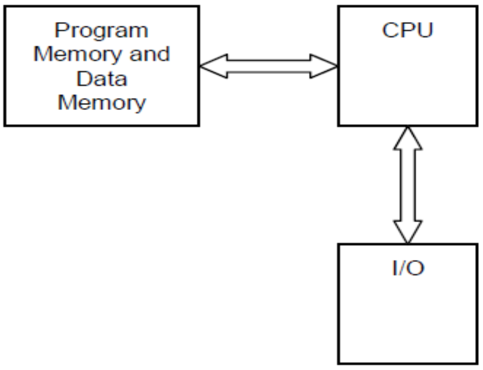
Cache Memory

- A cache is a small and very high speed memory used to speed up the transfer of data and instructions, doubling the speed of the computer in some cases.
- It can located inside or close to the CPU Chip
- it is placed between the CPU and the main memory.
- It stores frequently or most recently used instructions and data
- It is faster than RAM
- The data and instructions that are most recently or most frequently used by CPU are stored in cache memory.
- it is used to increase the speed of processing by making current programs and data available to the CPU at a rapid rate
- CPU processes data faster than main memory access time, thus processing speed is limited primarily by the speed of main memory.
- It compensates the speed difference between the main memory access time and processor logic.
- It is used to increase the speed of processing by making current programs and data available to the CPU at a rapid rate.
-
- The cache thus used for storing segments of programs currently being executed in the CPU and temporary data frequently needed in the present calculations
-
- The amount of cache memory is generally between 1kb and 512kb

How cache memory works

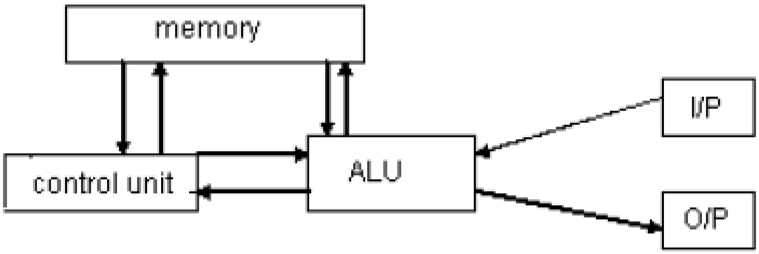


Von Neumann Architecture

Diagram A		**Diagram B**
 Von Neumann Architecture memory address bus CPU data bus both instructions and data go through here		

The Von Neumann Computer architecture has the following characteristics

- Stores both data and instructions in the **same** memory.
- Has a single processor which follows a **linear sequence** of the fetch-decode-execute cycle. Therefore it only processes one **job at a time with one set of data**. Execution occurs in a sequential fashion from one instruction to the next, unless explicitly modified.
- Uses serial processing of instructions. Allows for **one instruction to be rea**d from memory or data to be read/written from/to memory at a time.
- Instructions and data are stored in the same memory and **share a communication pathway or bus** to the CPU. Because program memory and data memory cannot be accessed at the same time, throughput is much smaller than the rate at which the CPU can work. This constraint (problem) is called the Von Neumann **bottleneck** and directly impacts the performance of the system. This can however be solved by use of cache memory.
- The von Neumann architecture allows instructions and data to be mixed and stored in the same memory module and the contents of this memory are addressable by location only. The execution occurs in a sequential fashion.
- Von Neumann architectures usually have a single unified cache
- The Von Neumann machine had five basic parts:- (i) Memory (ii) ALU (iii) control unit (iv) Input equipment & (v) output equipment: as shown below



- Processor needs two clock cycles to complete an instruction.

- In the first clock cycle the processor gets the instruction from memory and decodes it. In the next clock cycle the required data is taken from memory. For each instruction this cycle repeats and hence needs two cycles to complete an instruction
- Pipelining the instructions is not possible with this architecture.
- A **stored-program** digital computer is one that keeps its programmed instructions, as well as its data, in read-write, random access memory (RAM), that is the Von Neumann computer. This makes the machines much more flexible.

- By treating those instructions in the same way as data, a stored-program machine can easily change the program, and can do so under program control.
- Once in the computer’s memory a program will be executed one instruction at a time by repeatedly going through
- In the vast majority of modern computers, the same memory is used for both data and program instructions.

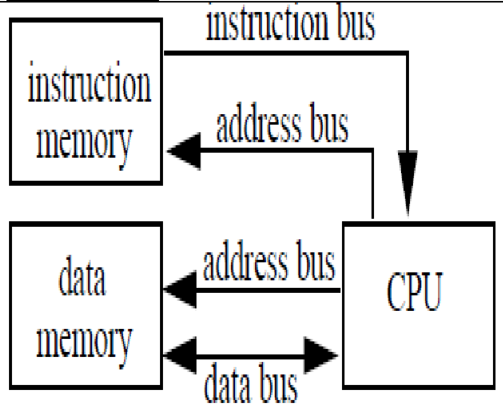
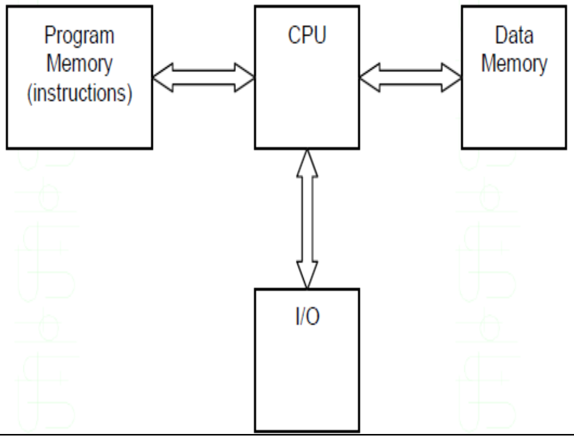
Advantages

- Almost all data can be processed by the von Neumann computer
- Cheaper than alternative types of processing
- Its design is very simple

Disadvantages

- Slower than other architectures
- Limited by bus transfer rate
- Does not maximise CPU utilisation
- Poorly written programs can have their data mixed up as both data and instructions share the same memory

<u>Harvard Architecture.</u>

Diagram A		**Diagram B**
		

The Harvard architecture has the following characteristics:

- Stores instructions and data in separate memory, thus has physically separate storage for data and instructions
- Has separate data and instruction buses, allowing transfers to be performed simultaneously on both buses.
- Data and instructions are treated separately
- May employ pipelining. Efficient Pipelining - Operand Fetch and Instruction Fetch can be overlapped.
- Harvard Architecture have a system would have separate caches for each bus.

Using a simple, unified memory system together with a Harvard architecture is highly inefficient. Unless it is possible to feed data into both busses at the same time, it might be better to use a von Neumann architecture processor.

Disadvantages

· Not widely used.
· More difficult to implement.
· More pins needed for buses.

System clock

- It is an electronic component that generates clock pulses to step the control unit through its operation.
- This sends out a sequence of timing pulses or signals, which are used to step the control unit through its operations.
- It generates electric signals at a fast speed
- It controls all functions of computer using clock ticks
- These ticks of system clock are known as clock cycle and speed of CPU
- The speed at which the CPU executes instructions is called clock speed or clock rate.
- It generates a continuous sequence of clock pulse to step the control unit through its operations

Serial Processing

Each instruction is executed in turn until the end of the program.

Advantages

- Nearly all programs can run on serial processing and therefore no additional complex code can be written.
- All data types are suitable for serial processing
- Program can use the previous result in the next operation
- Data set are independent of each other
- Cheaper to handle than parallel

Disadvantages

- Slows data processing especially in the Von Neumann architecture (bottleneck)
- Too much thrashing especially with poorly designed programs

Parallel Processing

- Parallel processing is the ability of a computer system to divide a job into many tasks which are executed simultaneously, using more than one processor, thus allowing multiple processing.
- Multiple CPUs can be used to carry out different parts of the fetch-execute cycle.
- The computer is able to perform concurrent data processing to achieve faster execution time.
- The system may have two or more ALUs and be able to execute two or more instructions at the same time.
- It may also have two or more processors operating concurrently
- The objective is to increase throughput
- Mostly applies to Single Instruction Single Data computer (SISD)
- Supercomputers utilizing parallel processing are used to maintain the safety.
- Scientists are using parallel processing to design computer-generated models of vehicles.
- Airlines use parallel processing to process customer information, forecast demand and decide what fares to charge.
- The medical community uses parallel processing supercomputers.

NB:-

- **Instruction Stream**:-the sequence of instructions read from memory
- **Data stream**: operations performed on the data in the processor

Parallel processing occurs in the instruction stream, the data stream or both.

(a) **Single Instruction Stream, Single Data Stream (SISD)** – Instructions are executed sequentially and parallel processing can be achieved by multiple functional units or by pipelining.

(b) **Single Instruction Stream, Multiple Data Stream (SIMD)-** includes multiple processing units with a single control unit. All processors receive the same instruction but operate on different data

(c) **Multiple Instruction Stream, Single Data Stream (MISD) –** Involves parallel computing where may functional units perform different operations by executing different instructions on the same data set

(d) **Multiple Instruction Stream, Multiple Data Stream (MIMD) –** processor capable of processing several programs at the same time

Advantages of parallel processing

- allows faster processing especially when handling large amounts of data
- more than one instruction (of a program) is processed at the same time
- Not limited (affected) by bus transfer rate
- Can make maximum CPU utilisation as long as it is kept full
- different processors can handle different tasks/parts of same job
- Memory is scalable with number of processors. Increase the number of processors and the size of memory increases proportionately.
- Each processor can rapidly access its own memory without interference and without the overhead incurred with trying to maintain cache

Disadvantages of parallel processing:

- Only certain types of data is appropriate for parallel processing
- Data that relies on previous operation cannot be made parallel.
- Each data set must be independent of each other
- Usually more expensive
- The programmer is responsible for the details associated with data communication.
- operating system is more complex to ensure synchronisation
- program has to be written in a suitable format
- Program is more difficult to test/write/debug
- It may be difficult to map existing data structures, based on global memory, to this memory organization.

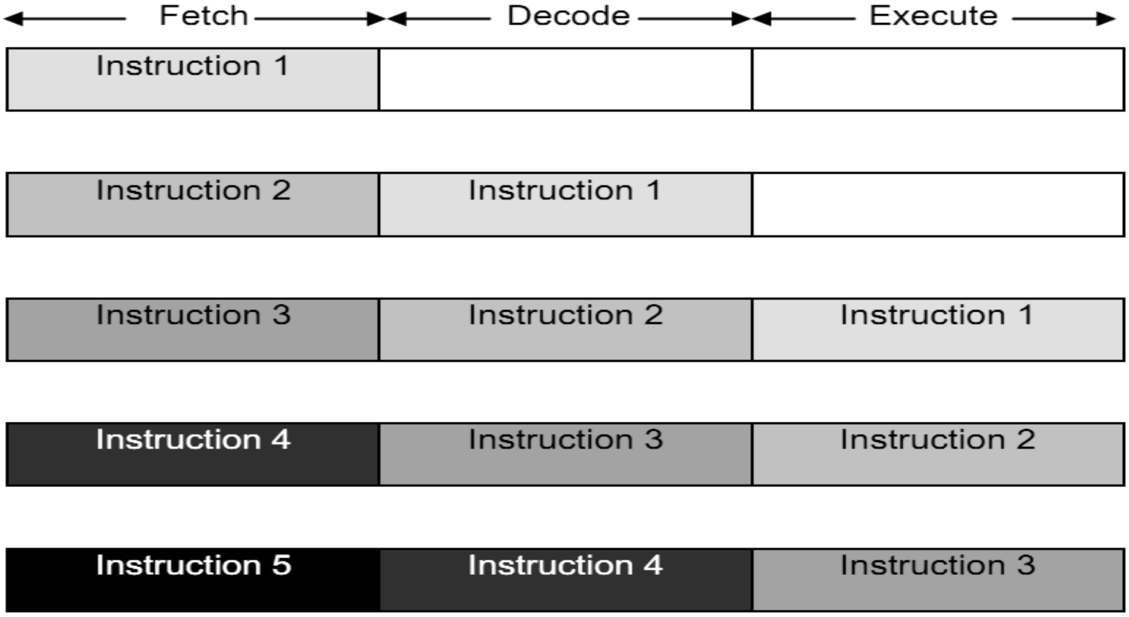
Parallel processing includes **Vector (Array) Processing** and **Pipeline Processing**
Involves the use of a several processors to perform a single job.

1. Array / Vector Processing

- It is when a processor allows the same instruction to operate simultaneously on multiple data locations and apply the same calculation on different data very fast on a processor with several ALUs
- Is a Single Instruction Multiple Data (SIMD)
- It applies to any example of a mathematical problem involving large number of similar calculations, e. g weather forecasting, airflow simulation around new aircraft, games, etc.
- It is a simple processor used for data that can be processed independently of one another –a single instruction can be applied to multiple *bits* of data all at the same time, and none of the results of that data will be needed to process the next bit of data.
- These types of processors are normally used for things like controlling input or output devices. They can be used to track the point of the mouse on a screen, or display the data on the monitor.
- They are sometimes called array processors, because the data is stored in an array, similar to that used in programing, the array can have one to many dimensions.
- Array processors are an extension to the CPU's arithmetic unit.
- The only disadvantage to array processing is that it relies on the data sets all acting on the same instruction, and it is impossible to use the results of one data set to process the next, although this does not matter in their use.
- It is also important to know, that an array processor is a single instruction multiple data (SIMD) processor, meaning that, lots of bits of data get processed with a single instruction.
- They are expensive to use it processing methods
- They have limitation on type of data to process

2. Pipeline Processing

- It is a technique which allows the overlapping of the fetch-decode-execute cycle for different instructions.
- A parallel processing architecture in which several processors are used, each one doing a different part of the fetch, decode, execute cycle, so the fetch-decode-execute cycle is staggered.
- The processor is split up into three parts (fetch, decode, execute), each of which handles one of the three stages.
- Each part is called a line, where each single line is a pipeline.
- This can be best illustrated with the diagram below.



Fetch	Decode	Execute
Instruction 1		
Instruction 2	Instruction 1	
Instruction 3	Instruction 2	Instruction 1
Instruction 4	Instruction 3	Instruction 2
Instruction 5	Instruction 4	Instruction 3

As long as the pipelines can be kept full, it is making best use of the CPU. This is an example of single instruction single data (SISD) processor, again it should be quite clear why, the processor is processing a single instruction to a single bit of data.

In pipelining, three instructions are dealt with at the same time. This reduces the execution time considerably.
However, this would only be true for a very linear program.
Once jump instructions are introduced the problem arises that the wrong instructions are in the pipeline waiting to be executed, so every time the sequence of instructions changes, the pipeline has to be cleared and the process started again.

A non-pipeline architecture is inefficient because some CPU components (modules) are idle while another module is active during the instruction cycle. Pipelining does not completely cancel out idle time in a CPU but making those modules work in parallel improves program execution significantly.

Processors with pipelining are organized inside into stages which can semi-independently work on separate jobs. Each stage is organized and linked into a 'chain' so each stage's output is fed to another stage until the job is done. This organization of the processor allows overall processing time to be significantly reduced

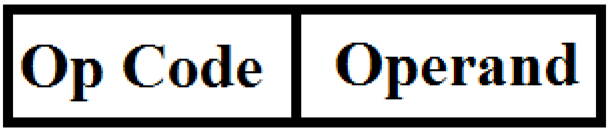
<u>Difference s between Vector and Serial Processors Program Codes</u>

Serial Processor	**Array/ Vector processor**
- execute this loop 10 times - read the next instruction and decode it - fetch this number - fetch that number - add them - put the result here - end loop	- read instruction and decode it - fetch these 10 numbers - fetch those 10 numbers - add them - put the results here

ADDRESSING MODES

Each instruction specifies an operation on certain data.
The different ways in which a computer calculate addresses holding the source and/or destination of the data being processed in a particular instruction is called addressing mode.

Addressing modes are mostly found in assembly language for microprocessors
Each assembly language instruction has the following structure:



Op Code	Operand

Op-code (operator):

- is the part that represent the operations that the computer can understand and carry out. It is the mnemonic part of the instruction/that indicates what it is to do/code for the operation. They are easier to remember. They can be represented by mnemonics which are the pseudo names given to the different operations that make it easier. E.g. ADD.

Operand:

- it is the address field in an instruction that holds data to be used by the operation given in the opcode, e.g. in ADD 12, "12" is the operand
- is the data to be manipulated, there's no point telling the computer what to ADD if there's no data to apply it to. It can hold the address of the data, or just the data.

The data is what the operation is being applied to, there are a number of different ways in which this data can be represented, and this is known as addressing.

Symbolic addressing: the use of characters to represent the address of a store location

Effective Address: the actual address of operand to be used by the instruction.

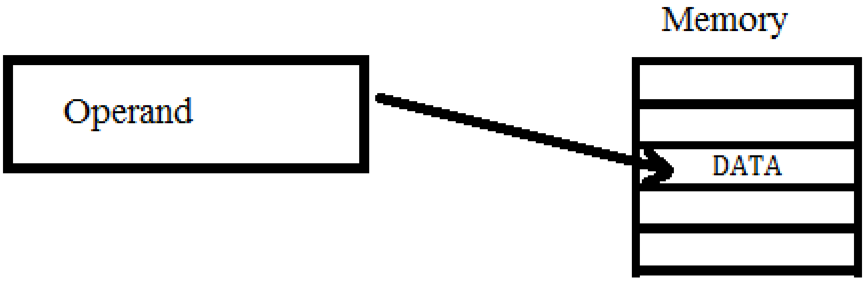
The most common addressing modes are: direct, indirect, indexed, relative and immediate addressing.

1. Immediate Addressing

- This is where the value to be used is stored in the instruction.
- This is when the value in the instruction is not an address at all but the actual data (constant to be used in the program).
- The data to be operated on is held as part of the instruction format.
- The data to be used is stored immediately after the op code for the instruction. Thus the operand field actually contains the data
- e.g: LDA #&80 : Means that Load the hexadecimal value of 80 into the accumulator register.
- MOVE #8, R1: Moves the value 8 into register R1
- Immediate addressing uses the # symbol.
- This is very simple, although not often used because the program parameters cannot be changed.
- This means that the data being operated on can't be adjusted and only uses constants.
- Can be used to initialize constants.

2. Direct Addressing

- The address in the instruction is the address to be used to get to the operand.
- The operand gives the address of the data to be used in the program.



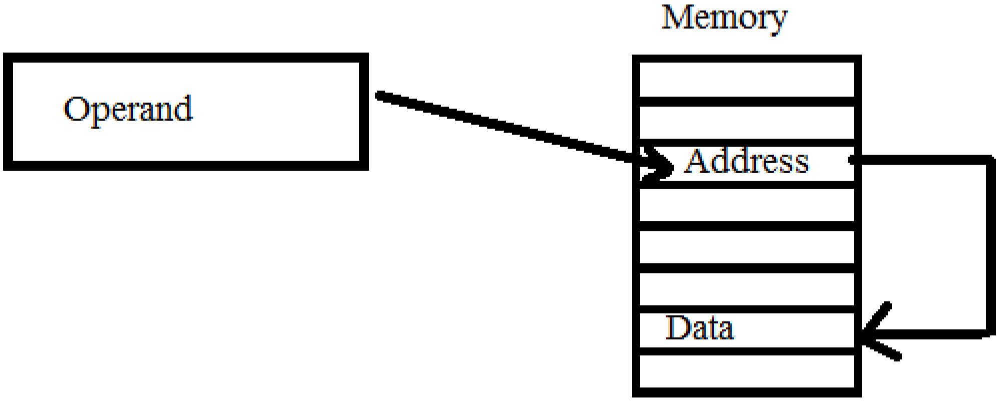
- It requires **one** memory reference to read the operand from the given location
- The address given in the instruction is the one that contains the data to be used in the operation **without any modification.**
- It is also called memory addressing
- e.g In the instruction **ADD 23**,
- we first go to memory address 23 which stores the instruction to be executed.
- It provides only a limited address space
- It is very simple, although does not make best use of memory
- It is slow as too much memory is used

Why is it not possible to use only direct addressing in assembly languages?

- Because the number of addresses available in memory is limited by the size
- the address field code is not re-locatable/code uses fixed memory locations

3. Indirect Addressing

- In this mode of addressing, the address given in the instruction holds the address of where the data is stored.
- This is whereby the real address is stored in the memory so the value in the address part of the instruction is pointing to the address of the data.
- The address of data in memory is held in another memory location and the operand of the instruction holds the address of this memory location.

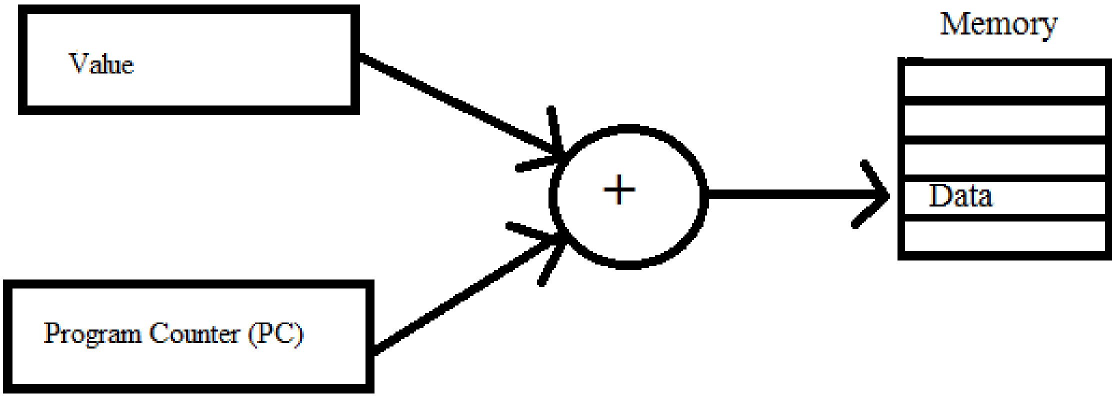


- It is MOSTLY used when access areas of memory that are not accessible using the space available for the address in the instruction code
- using our example, ADD 23
- we go to memory address 23
- there we are given another memory address, e.g 32, where the actual instruction will be stored

- This method is useful because the amount of space in a location is much bigger than the space in the address part of the instruction.
- It gives flexibility as the original program does not need to be altered if the position of the routines (sub-programs) change.
- Therefore we can store larger addresses and use more memory.
- It is used where memory larger than can be accessed by address in instruction
- It is also used when one wants to allow full size of register to be used for address
- used if memory locations are 32 bits are used and thus allowing more memory to be accessed
- there is a problem that some areas of memory cannot be addressed because size of memory address is larger than space available in instruction
- Indirect addressing solves this problem as the Memory address will fit in a memory location

Relative Addressing

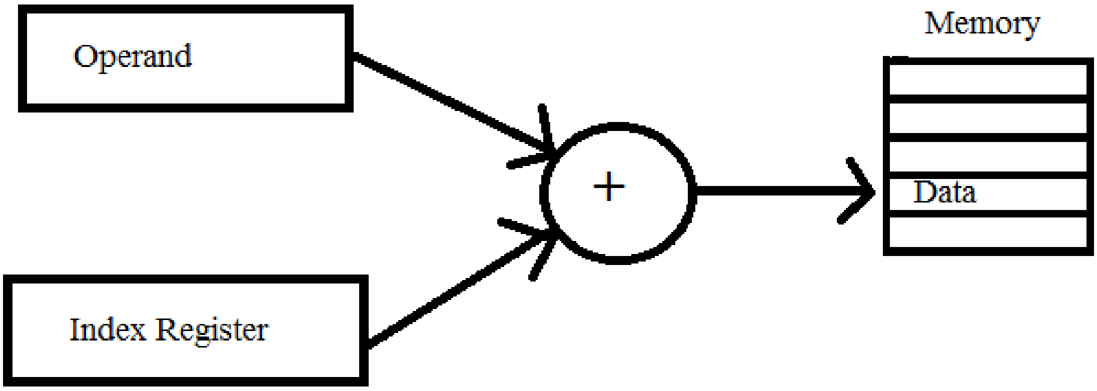
- The same as Indexed Addressng except that the PC replces the Index Register.
- E.g Load R_i, X (PC)
- This loads register R_i with the contents of the memory location whose address is the sum of the contents of the PC and the value X.



- This is direct addressing that does not commence from the start of the address of the memory.
- It begins from a fixed point, and all addresses are relative to that point.
- allows a real address to be calculated from a base address by adding the relative address
- relative address is an offset and can be used for arrays
- can be used for branching instructions
- it adds the PC contents to the base address to get the effective address
- it is appropriate when the code is going to be loaded at a random place in memory to be executed.
- Use to refer to jump instructions

Indexed Addressing

- The address part of the instruction is added to a value held in the index register.
- It is where the actual address is found by adding a displacement to the base address.



- The result of this is then the required address.
- The instructions specify two registers. The processor then adds contents of these two registers to get the effective address.
- One of the registers is an address register and it holds the base address.
- The other one is a data register or the displacement or index register
- Jmp +10: *branch to instruction 10 bytes on.*
- Used when a number of contiguous locations need to be accessed in order-e.g. contents of array
- Used **when** address in instruction does not change (need not to be altered-like constants), only contents of IR need to be changed

<u>Questions</u>

1. The Program Counter (Sequence Control Register) is a special register in the processor of a computer.

a) Describe the function of the *program counter*. (2)

b) Describe **two** ways in which the program counter can change during the normal execution of a program, explaining, in each case, how this change is initiated. (4)

c) Describe the initial state of the program counter before the running of the program. (2)

2. Explain what is meant by the term Von Neumann Architecture. (2)

3. Describe the fetch/decode part of the fetch/decode/execute/reset cycle, explaining the purpose of any special registers that you have mentioned. (7)

4. a) Describe how pipelining normally speeds up the processing done by a computer. (2)

b) State one type of instruction that would cause the pipeline system to be reset, explaining why such a reset is necessary. (3)

5). Give 3 differences between the Von Neumann and the Harvard Computer architectures.

CHAPTER 7: DATA STRUCTURES

A data structure is a collection of different data items that are stored together as a single unit and the operations allowable on them. Such data structures includes arrays, trees, linked lists, stacks and queues. Data structures can be static or dynamic.

Static data structure

- Are those which do not change in size while the program is running, e.g **arrays** and **fixed length records**.
- Most arrays are static, i.e., once you declare them, they cannot change in size.
- It main advantage is that amount of storage is known and therefore is easier to program

Advantages of Static Data Structures

- Easy to check for overflow
- Memory allocation is fixed and therefore there is no problem with adding and removing data items.
- Compiler can allocate space during compilation
- Easy to program as there is no need to check for data structure size at any given time/point
- An array allows random access and is faster to access elements than in other data structures

Disadvantages of Static Data Structures

- Programmer has to estimate maximum amount of space needed, which may be difficult
- Can waste a lot of space if some space is left with no data entered into it.
- Adding, removing and modifying elements is not directly possible. If done, it requires a lot of resources like memory.

DYNAMIC DATA STRUCTURES

- Can increase and decrease in size while the program is running, e.g. binary trees, linked lists, etc.
- they uses the space needed at any time, no limitations at all, unless computer memory is full
- its **size** changes as data is added & removed (size is not fixed)

Advantages of Dynamic Data Structures

- Only uses the space that is needed at any time
- Makes efficient use of the memory, no spaces lie idle at any given point
- Storage no-longer required can be returned to the system for other uses.
- Does not allow overflow
- There is effective use of resources as resources are allocated at run-time, as they are required.

Disadvantages of Dynamic Data Structures

- Complex and more difficult to program as software needs to keep track of its size and data item locations at all times
- Can be slow to implement searches
- A linked list only allows serial access

Static data structure

Array

An array is a static data structure, which stores and implements a set of items of the same data type using the same identifier name, in contiguous memory location. Every array element is accessed or referenced using an index (subscript), therefore uses index register addressing. It can either store integers, or names, but not both in one unit.

Declaration of arrays using VB 6.0

Arrays are declared by stating the following parameters: array name, array size, dimensions and the data type. The size of the array is the maximum number of elements that it can store. For example, the following declares an array that reserves **five** locations in memory and labels these as 'Names':

```
Dim Names(4) As String
```

This can also be declared as

```
Dim Names(0 to 4 ) As String
```

One may also declare the array as

```
Dim Names(1 to 5) As String
```

The last option modifies the starting value and the ending value of the indices, but the number of elements still remains the same.

By default, this implies that the array stores at most 5 elements. On running, the computer creates 5 contiguous memory locations under the name "Names". Thus Names will contain 5 partitions. Each memory partition will be accessed using an index, which is the address of the memory space in the array.

In Visual Basic, the number of elements stored in an array is N + 1, N being the number (subscript) used when declaring the array.

Array index starts at Zero up to N, N being the number in the declaration of the array.

The **index** is also called **Subscript**, and therefore arrays are subscripted data types.

It is not possible to exceed the upper limit or go below the lower limit on the array index values. You will receive a "Subscript out of range" error message if you try to do so.

In the above declaration, the array index starts from **0 to 4** and are **integer** values. The memory locations will be as follows:

Names[0]	Names[1]	Names[2]	Names[3]	Names[4]

The five individual locations are Names (0), Names (1), Names (2), Names (3) and Names (4).

Each data item is called an element of the array. To reference a particular element one must use the appropriate index.

NB: However, most programming languages differ with Microsoft Visual basic in handling arrays, especially on the amount of memory allocated. For example, using Java, the following declaration:

```
Int [4 ]Names;
```

This array declaration creates exactly 4 memory spaces for the array Names. The indices of the array range from 0 to 3 which are

Names[0], Names[1], Names[2] and Names[3]

Initialising an array in Visual Basic 6.0

The procedure of initializing an array in the computer memory is as follows:

- Size of array is calculated
- Location of array is decided according to data type and size
- Locations are reserved for the array
- Size of array is stored in a table
- Lower bound of the array is stored in a table
- Upper bound of array is stored in a table

- Data type is stored in a table
- Address of first element is stored in a table

<u>Inserting data into an array</u>

One may use the assignment statement or use looping structures. For example, the following statement assigns data to the 4th element:

Names(3) = “Kapondeni”

Arrays simplify the processing of similar data. An algorithm for getting four names from the user and storing them in the array Names is shown below:

```
Dim Names(4) As String
For i=0 to 4
        Input Value
        Names(i)=Value
Next i
```

One-dimensional arrays

A one-dimensional array is a data structure in which the array is declared using a single index and can be visually represented as a list.

The following diagram shows the visual representation of the array Names(4):

0	Theresa
1	Lameck
2	Johanne
3	Laurence
4	Fadzai

Two-dimensional arrays

A two-dimensional array is a data structure in which the array is declared using two indices and can be visually represented as a table.

Indices	0	1	2	3
0	Makombe	Tinashe	M	4A
1	Vheremu	Alex	M	4B
2	Mununi	Mary	F	3C
3	Chirongera	Salpicio	M	2C
4	Mutero	Violet	F	4C

The diagram above shows the visual representation of a 2 dimensional array Names(4,3)- 5 rows and 4 columns:

Each individual element can be referenced by its row and column indices. For example:

Names(0,0) is the data item “Makombe”
Names(2,1) is the item “Mary”
Names(1,2) is the item “M”

Initialising an array

Initialising an array is a procedure in which every value in the array is set with starting values – this starting value would typically be “” for a string array, or 0 for a numeric array.

Initialisation is important to ensure that the array does not contain results from a previous use, elsewhere in the program.

Algorithm for initialising a one-dimensional numeric array:

```
DIM TestScores(9) As Integer
DIM Index As Integer
FOR Index = 0 TO 9
    TestScores(Index) = 0
NEXT Index
```

Algorithm for initialising a two-dimensional string array:

```
DIM Students(4,3) As String
DIM RowIndex, ColumnIndex As Integer
FOR RowIndex = 0 TO 4
    FOR ColumnIndex = 0 TO 3
        Students(RowIndex,ColumnIndex) = “”
    NEXT ColumnIndex
NEXT RowIndex
```

Serial search on an array

The following pseudo-co de can be used to search an array to see if an item X exists:

```
01 DIM Index As Integer
02 DIM Flag As Boolean
03 Index = 0
04 Flag = False
05 Input X
06 REPEAT
07     IF TheArray(Index) = X THEN
08         Output Index
09         Flag = True
10     END IF
11     Index = Index + 1
12 UNTIL Flag = True OR Index > Maximum Size Of TheArray
13 IF Flag = False THEN
14     Show Message “Item not found”
15 END IF
```

Note that the variable Flag (line 04 and 09) is used to indicate when the item has been found and stop the loop repeating unnecessarily (line 12 ends the loop if Flag has been set to True).

The Actual Visual Basic Code will be as follows:

```
Dim Index As Integer
Dim Flag As Boolean
Dim x As Integer
Index = 0
 Flag = False
x = InputBox("Enter item to Search")
 Do
If Names(Index) = x Then
        MsgBox (Names(Index) & "  Has Been Found")
        Flag = True
    End If
    Index = Index + 1
 Loop Until (Flag = True Or Index > UBound(Names))
 If Flag = False Then
  MsgBox (x & " is not in the array")
 End If
```

Sorting Array Elements

```
Dim y As Integer
Dim p As Integer
Dim z As Integer
For y = LBound(Names) To UBound(Names)
    For p = LBound(Names) To UBound(Names) - 1
        If Names(y) < Names(p) Then
            z = Names(y)
            Names(y) = Names(p)
            Names(p) = z
        End If
    Next p
Next y
```

Deleting Array Elements

```
    Dim i As Integer
    For i = LBound(Names) To UBound(Names) - 1
        Names(i) = Names(i + 1)
    Next
    ' VB will convert this to 0 or to an empty string.
    Names(UBound(Names)) = Empty
```

The above algorithm will shift elements of the array up (or left), removing the first element and then completely removing the last element in the array.

The following is an alternative method of deleting an array element:

```
Dim Index As Integer
Dim Flag As Boolean
Dim x As Integer
Index = 0
 Flag = False
x = InputBox("Enter item to Search")
 Do
If Names(Index) = x Then
          MsgBox (Names(Index) & "  Has Been Found")
          Flag = True
          Names(Index) = Empty
     End If
     Index = Index + 1
 Loop Until (Flag = True Or Index > UBound(Names))
 If Flag = False Then
  MsgBox (x & " is not in the array")
 End If
```

The algorithm first searches the element to delete, and then remove it from the array.

NB:

- If the item is string, it replaces with empty spaces. However, if it is numeric, it replaces with a 0.
- Deleting form an array is often difficult as elements need to be shifted positions after deletion. It is an effective method for deleting elements.

Dynamic Declaration of Arrays:

Arrays can also be declared dynamically so that the user has to decide the size of the array when the program is running. The user does not need to specify the number of elements during the declaration stage. Dynamic declaration of arrays is done as follows:

```
Dim Names( ) As String
```

Inside the module, one then needs to redefine the array as follows:

```
ReDim Names(5) As String
```

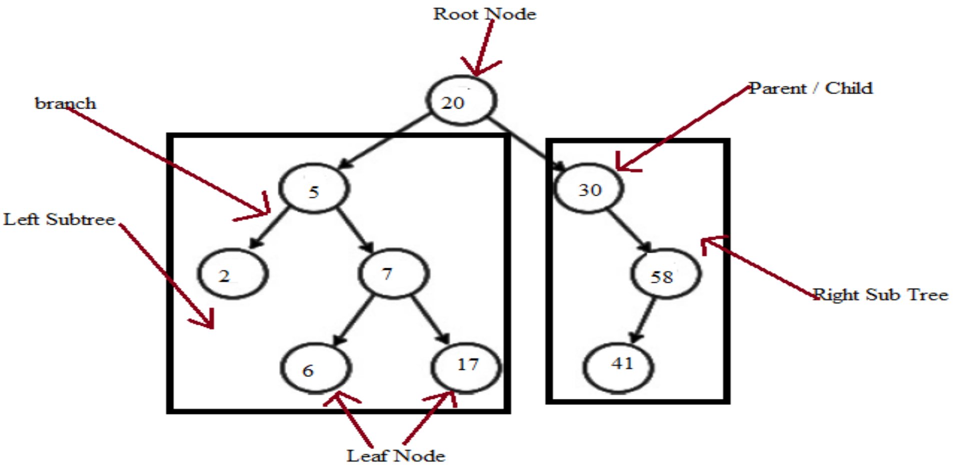
This allows the user to create the array when he/she actually needs it, using a *ReDim* statement: Dynamic arrays can be re-created at will, each time with a different number of items. When you re-create a dynamic array, its contents are reset to 0 (or to an empty string) and you lose the data it contains. If you want to resize an array without losing its contents, use the *ReDim Preserve* command:

ReDim Preserve Names(20) As String

DYNAMIC DATA STRUCTURE

Binary Trees

A binary tree is a data structure, consisting of a root node and zero, one or two sub-tree which are organised in a hierarchical way. Each node is a parent of at most two nodes.

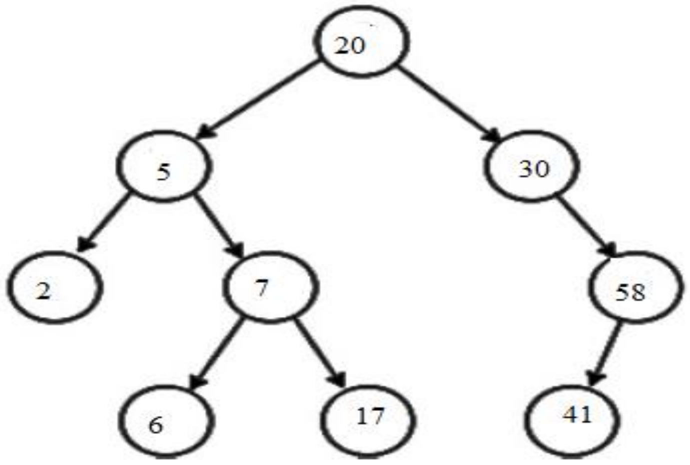


- The data items are held in **nodes**.
- The possible routes are called **paths/branches.** They are lines connecting the nodes.
- Each node has two possible paths.
- The nodes are arranged in layers.
- The first node is called the **root**, or **root node**. Each tree has only **one** root node. However, each branch can have its **branch root**.
- Node created by another one is called **child node (children)**
- Each child node has only one parent node
- Each parent node has at most two children
- The last node is called the **leaf node/terminal node** (has no children)

- Nodes that share common parent are called **siblings**

For example, given the following numbers: 20, 30, 5, 2, 7, 6, 17, 58, 41
Placing them in the binary tree is as follows:

- The first element becomes the root node, i.e. 20
- For other numbers, the bigger number goes to the right and the smaller one to the right of a node. Every time start from the root node, until you get to an empty space to place the new node.
- For example, 30, is bigger than 20, therefore is placed to the right hand side of 20. There is nothing on this side and therefore a new node is created and 30 placed inside.
- Next is 5, which is smaller than 20 (root node) and therefore goes to the left. There is an empty space therefore a new node is created and 5 is placed inside.
- Then 2 is smaller than 20 (root node) and therefore goes to the left. On the left there is 5. 2 is smaller than 5, therefore we go to the left and place 2 there.
- Next is 7, which is smaller than 20, we go to the left where there is 5. Seven (7) is bigger than 5, therefore we place it to the right of 5.
- ……….finish on your own!!!!!!!!!!!



Algorithm for constructing a binary tree.

```
Place first item into root node
For subsequent items, start from root node
Repeat
    If (new item > this node item) Then
        Follow right pointer
    Else
        Follow left pointer
    Endif
Until pointer =0
Place item at this node
```

<u>Binary tree traversal</u>

Tree traversal refers to means of walking through the tress structure such that each node is visited once. Traversal of trees is a recursive function-they call themselves. Common traversal methods are: Pre-Order, In-order and Post-Order Traversals.

A. Pre-Order traversal

The order of traversal is:

- Visit the Node
- Traverse the Left sub-tree

- Traverse the Right sub-tree.

This is generally given as **NLR**

For the diagram above, the pre-order traversal will be as follows:

20, 5, 2, 7, 6, 17, 30, 58, 41.

The algorithm for pre-order traversal is as follows:

1. *Print current node*
2. *For the current node, check if there is left sub-tree*
3. *If there is left sub-tree, go to the root node of this sub-tree and print it*
4. *For the current node, check if there is right sub-tree*
5. *If right sub-tree is present, then go to 6,* ***else*** *go to 7*
6. *Repeat 1*
7. *End*

Recursive Algorithm for pre-order traversal

```
Procedure traversefrom(p)
    Print (data);
    If Tree[p].Left<>0 Then
        Traversefrom(Left)
    EndIf
    If Tree(p).Right <> 0 Then
        traversefrom(Right)
    endif
endProcedure
```

B. In-Order Traversal

The order of traversal is:

- Traverse the Left sub-tree
- Visit the Node
- Traverse the Right sub-tree.

This is generally given as **LNR**

For the diagram above, the pre-order traversal will be as follows:

2, 5, 6, 7, 17, 20, 30, 41, 58.

<u>NB</u>: In-order traversal prints items in **ascending order** or in **alphabetical order** if they are alphabetic items.

In-order traversal algorithm:

1. *For the current node, check if there is left-sb-tree. If it exists, go to the root node of this sub-tree and then go to 2. If it doesn't exist, go to 3.*
2. *Repeat 1*
3. *Print the current node*
4. *For the current node. Check whether it has a right sub-tree. If it has, go to 5. Else go to 6*
5. *Repeat 1*
6. *End*

Recursive Algorithm for in-order traversal

```
Procedure traversefrom(p)
    If Tree[p].Left<>0 Then
        Traversefrom(Left)
    EndIf
    Print (data);
    If Tree(p).Right <> 0 Then
```

```
                    traversefrom(Right)
                endif
        endProcedure
```

C. Post-Order Traversal

The order of traversal is:

- Traverse the Left sub-tree
- Traverse the Right sub-tree.
- Visit the Node

This is generally given as **LRN**
For the diagram above, the pre-order traversal will be as follows:

2, 6, 17, 7, 5, 41, 58, 30, 20.

The algorithm for post-order traversal is as follows:

```
        Procedure traversefrom(p)
                If Tree[p].Left<>0 Then
                        Traversefrom(Left)
                EndIf
                If Tree(p).Right <> 0 Then
                        traversefrom(Right)
                endif
                Print (data);
endProcedure
```

Inserting Data into a Binary Tree

- *Look at each node starting from the root node*
- *If the root is empty, create the node and place the value*
- *If the new value is less than the value of the of the node, move left, other wise move right*
- *Repeat this for each node arrived until there is no node*
- *Then create a new node and insert the data.*
- *Perform until no new item need to be added*

This can be written algorithmically as:

1. If tree is empty, enter data item at root and stop.
2. Current node = root.
3. Repeat steps 4 and 5 until current node is null.
4. If new data item is less than value at current node go left else go right.
5. Current node = node reached (null if no node).
6. if node reached is null, create new node and enter data.

This can also be written as:

```
    Repeat
        Compare new value with root value
        If new value > root value then
            Follow right sub-tree
        Else
            Follow left sub-tree
        Endif
    Until no sub-tree
    Insert new value as root of new sub-tree.
```

Deleting Data from a Tree

Deleting data from a tree is quite complicated, because if it has sub-nodes, these will also be deleted. There are two options however:

1. The structure could be **left the same**, but the value of that node **set to deleted**.
2. The tree could be traversed, the **values removed**, put into a stack and then put back into a binary tree.

Problems when deleting element from tree and solution

- The value at the node is not only data, it is part of the structure of the tree
- If the node is simply deleted then the sub-tree leading from it is not navigable

*__NB__: the algorithm to delete a leaf node is straightforward as deleting a leaf node does not change the structure of the tree

To remove the value from the tree, either:

- It remains in the tree structure
- Mark value as deleted so that it cannot be output but acts as the root for its sub-tree

OR:

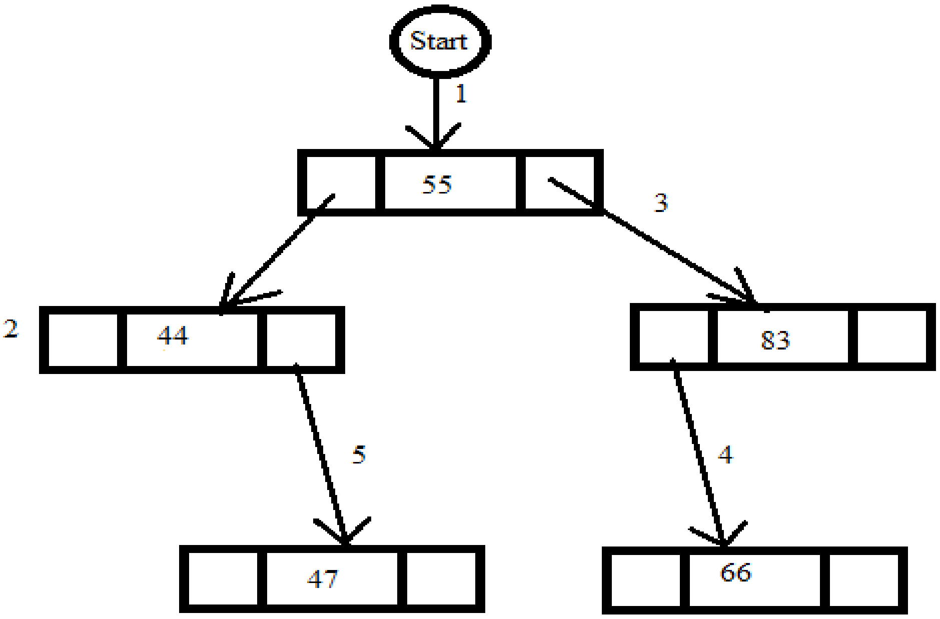
- The entire sub-tree without its root is read to a list
- The sub-tree is deleted
- The values in the list are read back into the tree (element which was originally on the left will replace the deleted element (becomes root of that branch))

Searching item from Binary Tree

The algorithm is as follows:

```
Enter item to search(this item)
Start at root node
Repeat
    If wanted item = this item Then
        Found = True
        Display Item
    Else
        If wanted Item >this item Then
            Follow right pointer
        Else
            Follow left pointer
        EndIf
    EndIf
Until (Found=True or Null pointer is encountered)
```

Binary tree maintain the order of elements. However, if one element is deleted, the order is affected. In some cases, each node (data) may be assigned pointers (right and left pointer). This is as illustrated below:

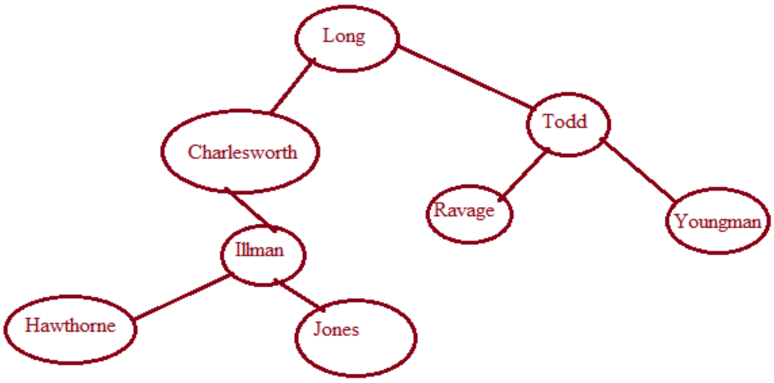


IMPLEMENTATION OF BINARY TREES USING ARRAYS

Binary trees can be implemented using left and right pointers for each node. Each node will have the following:

- Left pointer
- Right pointer
- Data item

Let's take the binary tree below:



These can be illustrated as follows:

	Left	Data	Right
Tree [1]	2	Long	5
Tree [2]	0	Charlesworth	3
Tree [3]	4	Illman	7
Tree [4]	0	Hawthorne	0
Tree [5]	8	Todd	6
Tree [6]	0	Youngman	0
Tree [7]	0	Jones	0
Tree [8]	0	Ravage	0

- The pointer value 0 indicates a 'nil' pointer, thus a node will be pointing to nothing.
- Tree[1].Left = 2
- Tree[Tree[1].Left].Right = 3
- Tree[6].Data = "Youngman"

Binary trees are also important in the evaluation of postfix expressions (Reverse Polish Notation)

Infix Expressions

Normal mathematical expressions are written as follows:

A + B

This method is called **Infix Notation**, because the operator is found between the operands to be acted upon. Infix notation involves the use of brackets and observes operator precedence, e.g. BODMAS/BOMDAS.

For example, the expression below is an Infix:

(A+B)*C+(D-A)

If there are no brackets, the expression will give a different answer. It is not easy for the computer to evaluate infix expressions.

Prefix Expressions

The **Polish Notation** is also called the **prefix**. In Prefix (Polish) notation, the operator precedes the operands. For example, the infix expression

A + B is given as: **+AB**

This has an advantage that it removes ambiguity and avoids use of brackets

Postfix Expressions

Reverse Polish Notation (Postfix) is a way of writing mathematical expressions without using parenthesis and brackets, and in which the operands precede the operator. **Reverse Polish** Notation is also called **Postfix**. For example, the Infix expression A + B is written as follows in Postfix:

AB+

Likewise, the infix expression (A+B)*C+(D-A) is written as:

AB+C*DA-+

There is no need for brackets in this situation.

Reverse Polish Notation has the following benefits:

- can be processed directly by reading the expression from left to right
- is free of ambiguities
- does not require brackets
- does not require rules of precedence as in BODMAS/BOMDAS
- can be processed using a stack

Conversion of Infix Expression into Reverse Polish Notation

This can be easily done using the binary tree. The expression is written down as a binary tree, then a post-order traversal is undertaken, writing down the nodes visited. The result will be a Reverse Polish Notation of the original expression.

The first procedure is to put elements into a binary tree, with each operator or operand being a node on its own.

To do this, we look at operator **precedence**. The operator with the **lowest / weakest** precedence becomes the root node. When the weakest operator is identified, it is placed as the root node.

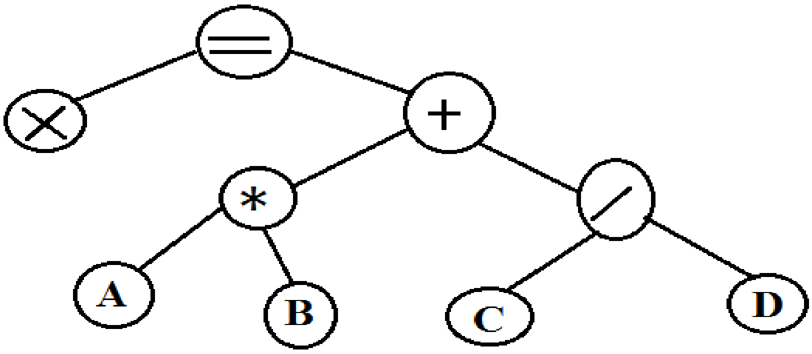
Expressions on the left hand side of this weakest operator makes up the left sub tree, while those on the right hand side makes the right sub-tree. This procedure is applied as we move down the tree.

Let's take an example:

X = A * B + C / D

- The weakest operator is the = sign. So it becomes the **root** node.

- To its left, there is only X, which becomes a node to the left of =
- There are no more items to the left branch of the root node. We therefore go to the right side.
- The weakest operator to the right of = is +, therefore it becomes the first node to the left of the root.
- To the left of the + sign, the weakest sign is the *, which we place as the node to the left of +
- To the left of * there is A, which becomes a node.
- To the right of * there is B, which becomes a node.
- We are now done with the left branch of the + node, lets move to the right side.
- The weakest sign is /, it becomes a node there.
- To its left there is C, which becomes a node.
- To the right of / there is D, which also becomes a node.
- Thus all the items are now in a binary tree , which appears as follows:

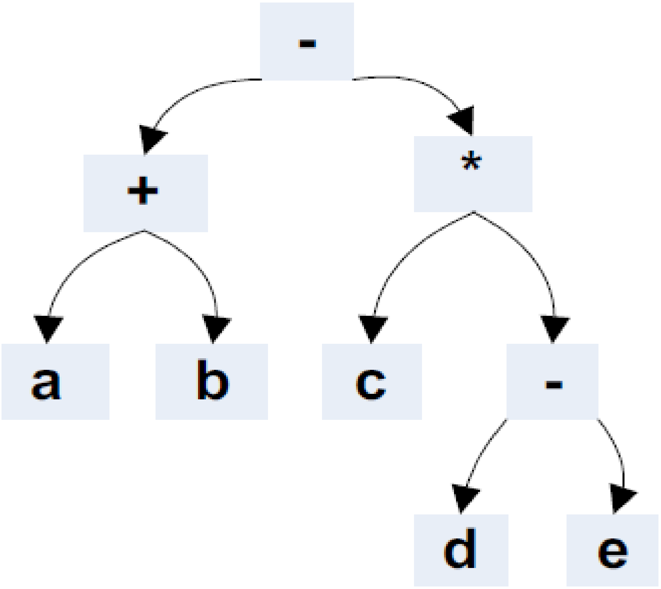


- Now, visiting each node in post-order we get:
  X A B * C D / + =

*which is the final **Reverse Polish Notation**.*

<u>NB</u>: If the original expression to be converted contains, brackets or parenthesis, ignore them, that is, don't put them in the tree, but use them in getting the weakest operator. Items in brackets have a higher priority and therefore are inserted in the tree latter than those **<u>NOT</u>** in brackets.

Given the following infix notation: (a+b) – c*(d-e)
This can be represented using a tree as follows:



Using the above tree, the reverse polish notation will be as follows;
a b + c d e - * -

Evaluating Reverse Polish expressions

A **Reverse Polish** expression is evaluated using a stack.

- Any operand encountered is loaded into a stack (that is a number or variable)
- If the next symbol is an operator (+, -,*,…), the last two operands in the stack are popped,
- The first popped item is placed to the right which the last popped one is placed to the left, and the operator place in on the middle.
- The operation is carried out and the result is pushed back into the stack.

The procedure above can be shown diagrammatically as below:

Let us assume that the following postfix expression:

*a b + c d e - * -*

with the following values. A = 5, b=3, c=2, d = 6 and e=4.

- The operands in this expression are: a, b, c, d and e.
- Operators are +, - and *

- First a (5) is pushed into the stack
- b (3) is pushed into the stack
- an operator is encountered, therefore, b (3) is popped and placed on the right
- a (5) is popped and placed on the left
- the operator (+) is placed in between and effected
- the result (8)is pushed back into the stack
- c (2) is pushed into the stack
- d (6) is pushed into the stack
- e (4) is pushed into the stack
- - sign is encountered. Thus e (4) is popped and placed on the right
- e (6) is popped and placed on the left, to give (d-e)
- Operator (-) is effected
- The result (2) is pushed back into the stack.
- Another operator is encountered(*)
- Result (2) is popped
- C (2) is popped and operator effected
- The result (4) is pushed back to the stack.
- Operator (-) again is encountered
- Result(4) is popped
- Another result (8) is popped and operator effected
- Result (4) is pushed into the stack
- There are no more operators. The answer is displayed.

The above can be shown diagrammatically. Can please draw this on your own?

Questions

1. (a) State the difference between dynamic and static data structures giving an example of each. (3)
b) Show how a binary tree can be used to store the data items Feddi, Eda, Joh, Sean, Dav, Gali in alphabetic order. (4)
c) Explain why problems may arise if Joh is deleted from the tree and how such problems may be overcome.

2. An array is to be used to store information. State three parameters that need to be given about the array before it can be used, explaining the reason why each is necessary.

3. (a) Explain the difference between static and dynamic data structures. [2]

(b) Give an example of a

(i) static,

(ii) dynamic

data structure, giving an advantage of each. [4]

(c) The details of a car part are stored in a binary tree according to this algorithm

```
READ VALUE NEW_PART
START AT ROOT NODE
WHILE NODE NOT EMPTY, DO
IF NEW_PART < VALUE AT NODE
THEN FOLLOW LEFT SUBTREE
ELSE FOLLOW RIGHT SUBTREE
ENDIF
ENDWHILE
INSERT NEW_PART AT NODE
END
```

(i) Show the binary tree after the following values have been input

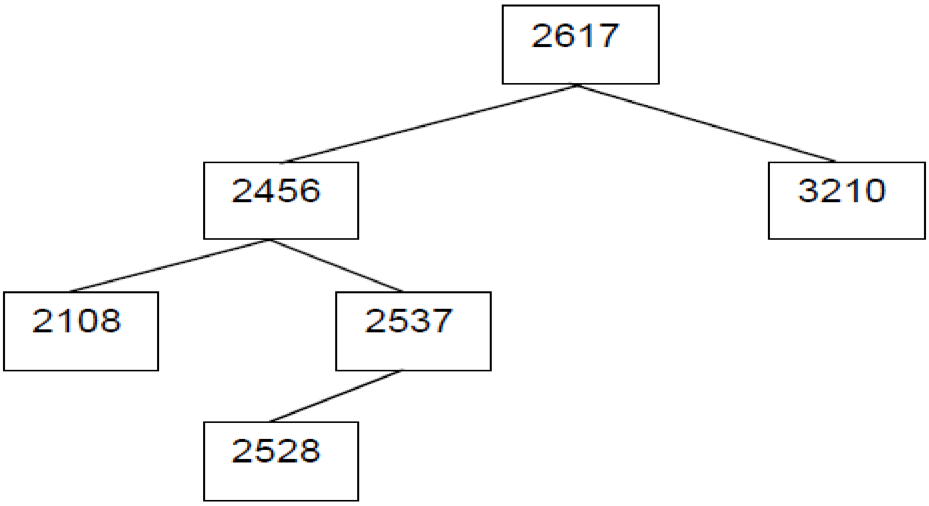
Radio Visor Brakes Tyres Alternator Windscreen [3]

(ii) Explain how Clutch is added to the tree in (i). [5]

(iii) Describe an algorithm that can be applied to the binary tree of car parts, so that the tree is read in alphabetic order.

4.

The following binary tree diagram contains a number of integers. In each case the right pointer indicates the condition ***"higher number"*** and the left pointer indicates the condition "***lower or equal number".***



i) Write down the integer contained in the root node. [1]

(ii) Write down the order in which the nodes would be accessed to find the integer 2528. [1]

(iii) Copy the tree and show where a new node containing the integer 3106 would be added. [1]

(iv) The integer 2550 is not in the diagram. Explain what would happen if a search was made for this code.

5. An array x[1..n] contains a number of integer values.

Design an algorithm (in pseudo-code or a high- level programming language) which will carry out a linear search of the array for a given search element. If the element is found, it should output the position of the search element. If the element is not found, the algorithm should output zero.

CHAPTER 8: LOGIC GATES

- A logic gate is a device that produce signals of 1 or 0 when the input logic requirements are met and are used in manipulating binary information.
- A logic gate is a device (or electrical circuit) that performs one or more logical operations on one or more input signals.
- Its output represent **Boolean** (T or F) or binary values (1 or 0) as voltages.
- Logic gates are the building blocks of digital technology.
- They can be used in applications like:
  - Building computer chips
  - Programming traffic signals
  - Chips for automatic alarm systems
  - Chips for automated control systems

- Electronic circuits operate using binary logic gates.
- Logic gates process signals which represent **TRUE** or **FALSE, ON** or **OFF** , **1** or **0**

Main Logic Gates

The main logic gates are:

(a) OR gate
(b) AND gate
(c) NOT gate
(d) NOR gate
(e) NAND gate
(f) Exclusive OR gate (XOR)
(g) Exclusive NOR gate (XNOR)

Logic gates are used with **truth tables**.

- A ***truth table*** is a table which shows how a *logic* circuit's output responds to various combinations of the inputs, using ***logic*** **1** for true and ***logic*** **0** for false.
- A truth table is a table that describes the behaviour of a logic gate.
- It lists the value of the output for every possible combination of the inputs
- Truth tables contains 1s and 0s and are an integral part of logic gates functionality.
- Truth table and logic gates use the following:
- 1 (True, ON, Not False)
- 0 (False, OFF, Not True)

The number of rows in a truth table shows the number of combinations of the inputs of a particular circuit. The number of rows for each gate is found using the following formulae: rows = 2^n , *n* being the number of inputs in the gate or circuit. For example, a gate or circuit has the following rows corresponding to the number of input (excluding column headings):

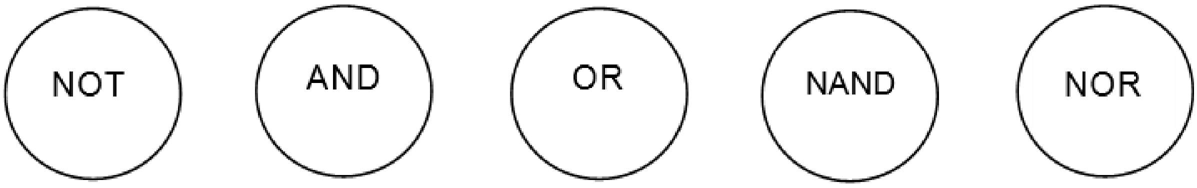
- 1 input = 2^1 = 2 rows
- 2 inputs = 2^2 = 4 rows
- 3 inputs = 2^3 = 8 rows
- …..

Graphical Representation of Gates and their Truth Tables

Each logic gate has its own unique graphical representation, which can be in general form or in standard form.

(1) General form

Each logic gate has a circle and the name of the gate to differentiate it from the rest as given below:

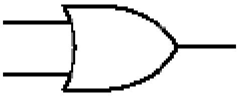


The name inside the gate gives us the type of the gate

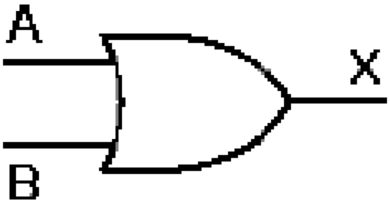
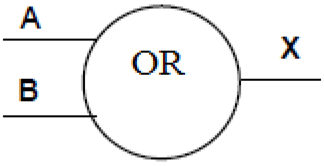
(2) Standard Representation

In standard form, each logic gate has its own unique diagram. Even if the name of the gate is not written, one knows what it stands for because of the shape. The following are the logic gates and their shapes in standard form.

(a) OR gate



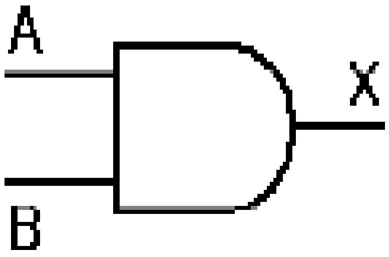
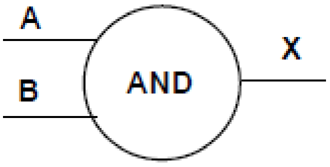
This represents two inputs entering the gate and one output from the gate. The inputs can be represented by **any** alphabetic characters, e.g. A and B, while the output can be X, given as follows:

<u>Logic Gate Diagram</u>		**<u>Truth table</u>**
Standard Form	**General Form**	
		A B X 0 0 0 0 1 1 1 0 1 1 1 1

- X= A **OR** B
- The output (X) is **true** if the **INPUT A OR INPUT B** are **true.**
- Thus if any one of the inputs is 1, the output is automatically 1
- Output only becomes 0 if all inputs are 0

(b) AND gate

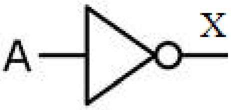
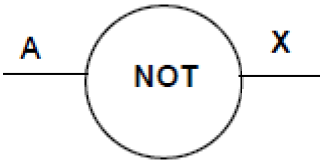
This is represented as follows:

Logic Gate Diagram		**Truth table**
Standard Form	**General Form**	
		

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

The output (X) is only **true** if the **INPUT A AND INPUT B** are both **true**. If any one of the inputs is 0, then the output becomes 0 also.
Thus X = A **AND** B.

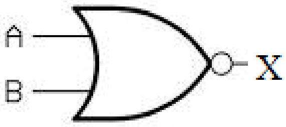
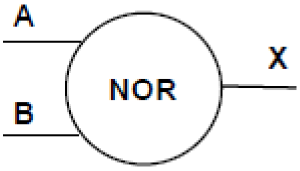
(c) NOT gate

Logic Gate Diagram		**Truth table**
Standard Form	**General Form**	
		

INPUT A	OUTPUT X
1	0
0	1

The **NOT** gate has **only** one input and one output. The input is negated. Thus if input is 1, output is 0, and vice versa.
The output (X) is **true** when the **INPUT A** is **NOT TRUE**.
The output (X) is **False** when the **INPUT A** is **TRUE.**

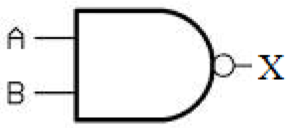
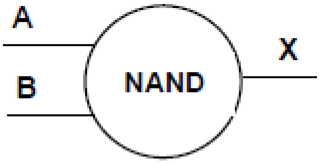
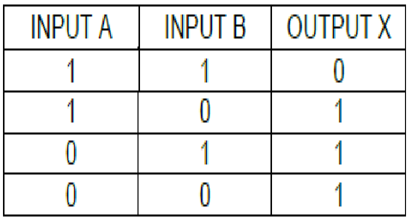
(d) NOR gate

Logic Gate Diagram		**Truth table**
Standard Form	**General Form**	
		

INPUT A	INPUT B	OUTPUT X
1	1	0
1	0	0
0	1	0
0	0	1

- This is an **OR** gate with the output X inverted.
- The output (X) is **true** if **NOT** (**INPUT A OR INPUT B**) are **true.**
- Thus X = NOT (A or B)

(e) NAND gate

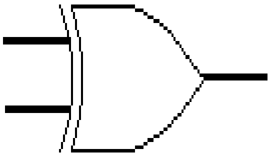
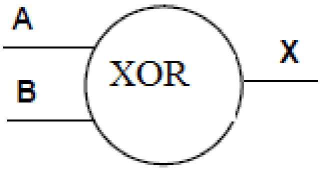
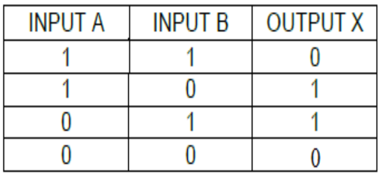
Logic Gate Diagram		**Truth table**
Standard Form	**General Form**	
		

INPUT A	INPUT B	OUTPUT X
1	1	0
1	0	1
0	1	1
0	0	1

This is an **AND** gate with the output X inverted.
The output is **true if INPUT A AND INPUT B** are **NOT** both **True.**
It translates to NOT (A and B)

(f) Exclusive OR gate (XOR)

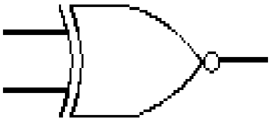
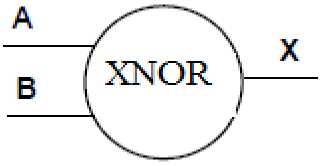
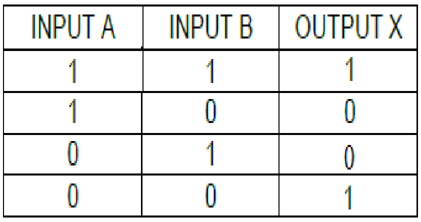
In this gate, the output is 1 (T) if either, but not both, of the inputs are 1 (T). The output is 0 (False) if both inputs are 0(False) or if both inputs are 1(True). In other words, the output is 1 if the inputs are different, but 0 if the inputs are the same.

Logic Gate Diagram		**Truth table**
Standard Form	**General Form**	
		

INPUT A	INPUT B	OUTPUT X
1	1	0
1	0	1
0	1	1
0	0	0

(g) Exclusive NOR Gate (XNOR)

The *XNOR (exclusive-NOR) gate* is a combination XOR gate followed by an inverter. Its output is 1 if the inputs are the same, and 0 if the inputs are different.

Logic Gate Diagram		**Truth table**
Standard Form	**General Form**	
		

INPUT A	INPUT B	OUTPUT X
1	1	1
1	0	0
0	1	0
0	0	1

Combinational logic gates

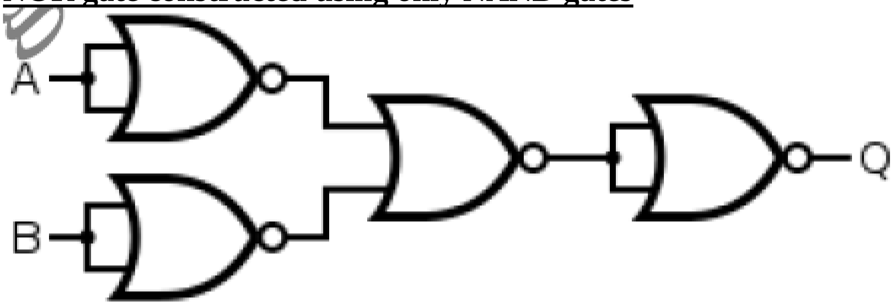
A combination of logic gates, which may be different, gives a logic circuit as given below:

BOOLEAN EXPRESSIONS

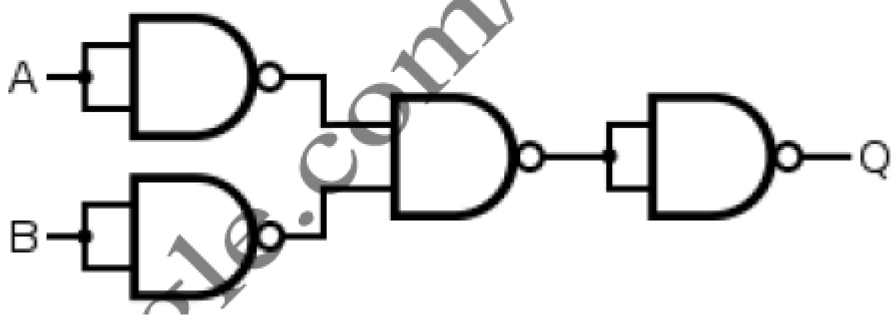
Boolean Expressions are equivalent expressions of the logic state of gates. For example, the Boolean expression for:
a **NOT** gate with input A and output C: C = **NOT** A

NAND and NOR gates are known as **universal gates** because they are inexpensive to manufacture and any Boolean function (AND, OR, NOT) can be constructed using only NAND or only NOR gates. Even NAND and NOR gates can be used as each other's alternatives in a circuit.

NOR gate constructed using only NAND gates



NAND gate constructed using only NOR gates



Expressions and logic gates

Expressions using mathematical symbols can be used to represent logic gates.
One may be required to draw logic gates using such mathematical expressions
Such symbols and their meaning are as given below:

1. **Plus sign (+)**
   - This means OR, e.g.
   - C= A + B
   - This is an OR gate, which means C = A OR B.

2. **Multiplication Sign**
   - This represents an AND gate
   - E.g C = AB,
   - C = AxB,
   - C = A.B
   - C= (AB)(AC)
   - C = (A.B).(A.C)
   - All these are various versions of the AND gate.

3. **Complement sign (‘ or – above an input)**
   - This represents a NOT gate
   - For example: NOT A can be given as any one of the following:
     - A’
     - $\overline{A}$

Logic gate Problems:

Question: Worked Example

A steel rolling mill is to be controlled by a logic network made up of AND, OR and NOT gates only. The mill receives a stop signal (i.e. S = 1) depending on the following input bits:

INPUT	BINARY VALUE	CONDITION
L	1 0	Length > 100 metres Length ≤ 100 metres
T	1 0	Temperature < 1000 C Temperature ≤ 1000 C
V	1 0	Velocity > 10 m/s Velocity ≤ 10 m/s

A stop signal (S = 1) occurs when:
either: Length, L > 100 metres and Velocity, V < =10 m/s
Or Temperature, T <=1000 C and Velocity, V >10 m/s

Draw a logic network and truth table to show all the possible situations when the stop signal could be received.

Answer

Step 1: Deduce and Write the logic statement

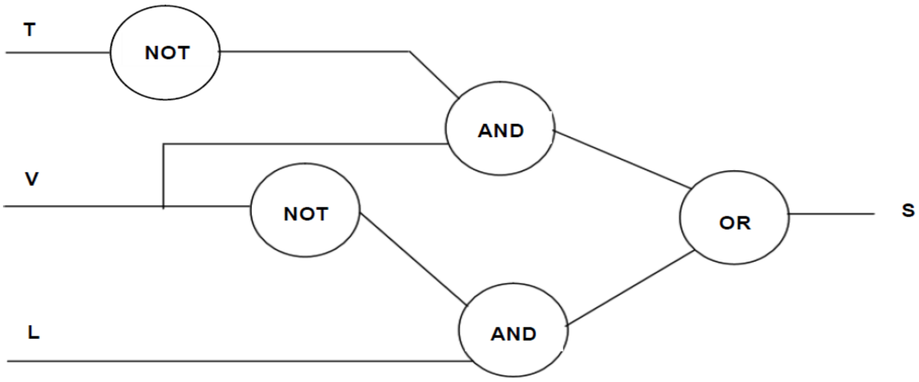
- The first statement can be re-written as: **(L = 1 AND V = NOT 1)** since Length > 100 metres corresponds to a binary value of 1 and Velocity <=10 m/s corresponds to a binary value of 0 (i.e. NOT 1).
- The second statement can be written as (T = NOT 1 AND V = 1)
- Both these statements are joined together by OR which gives us the logic statement: if (L = 1 AND V = NOT 1) OR (T = NOT 1 AND V = 1) then S = 1
- The above statement can be written as: S = 1 if (L = 1 AND V = NOT 1) OR (T = NOT 1 AND V = 1)

NB: the Student should first of all write the following logic statement before coming up with a truth table or logic circuit as this has some marks awarded to it., i.e.

S = 1 if (L = 1 AND V = NOT 1) OR (T = NOT 1 AND V = 1)

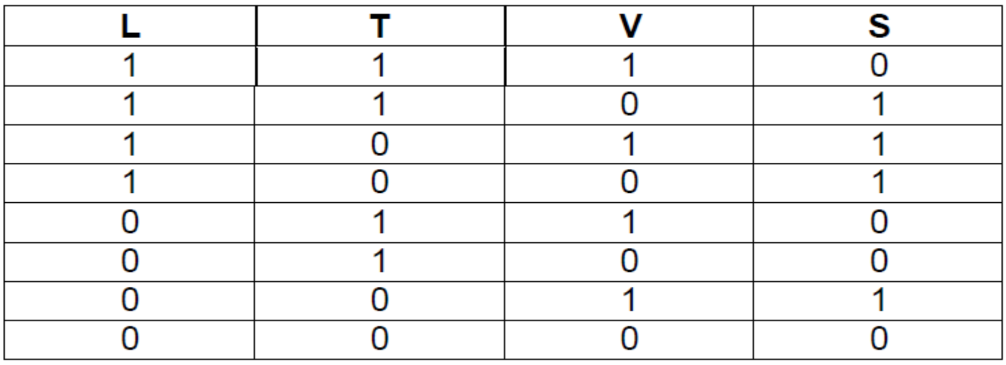
Step 2: Logic Circuit

- Using the logic statement above, one can now draw the logic circuit as given below:



Step 3: Truth Table

- One can now draw the truth table, basing from the logic statement in Step 1.



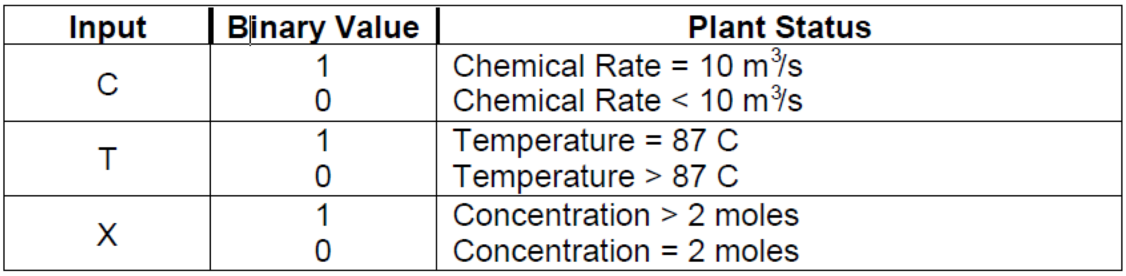
L	T	V	S
1	1	1	0
1	1	0	1
1	0	1	1
1	0	0	1
0	1	1	0
0	1	0	0
0	0	1	1
0	0	0	0

Questions

1. A computer will only operate if three switches P, S and T are correctly set. An output signal (X = 1) will occur if R and S are both ON or if R is OFF and S and T are ON. Design a logic network and draw the truth table for this network.

2. A traffic signal system will only operate if it receives an output signal (D = 1).
This can only occur if:
Either (a) signal A is red (i.e. A = 0)
Or (b) signal A is green (i.e. A = 1) and signals B and C are both red (i.e. B and C are both 0)
Design a logic network and draw a truth table for the above system.

3. A chemical plant gives out a warning signal (W = 1) when the process goes wrong. A logic network is used to provide input and to decide whether or not W = 1

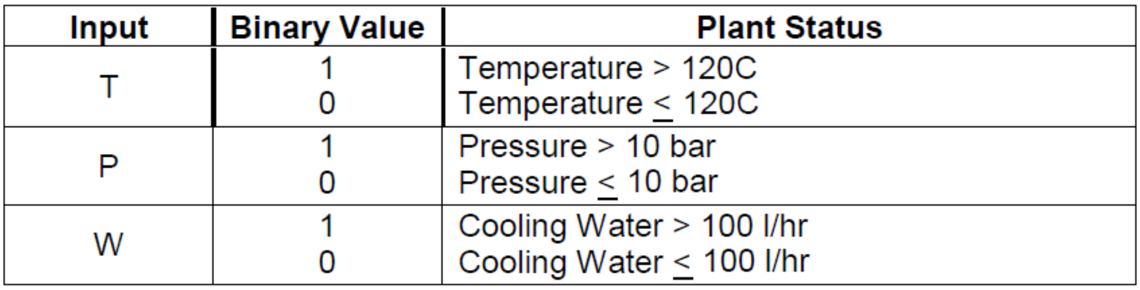


Input	Binary Value	Plant Status
C	1 0	Chemical Rate = 10 m^3/s Chemical Rate < 10 m^3/s
T	1 0	Temperature = 87 C Temperature > 87 C
X	1 0	Concentration > 2 moles Concentration = 2 moles

A warning signal (W = 1) will be generated if
either (a) Chemical Rate < 10 m /s
or (b) Temperature > 87 C and Concentration > 2 moles
or (c) Chemical rate = 10 m /s and Temperature > 87 C

Draw a logic network and truth table to show all the possible situations when the warning signal could be received

4. A power station has a safety system based on three inputs to a logic network. A warning signal (S = 1) is produced when certain conditions occur based on these 3 inputs:



Input	Binary Value	Plant Status
T	1 0	Temperature > 120C Temperature ≤ 120C
P	1 0	Pressure > 10 bar Pressure ≤ 10 bar
W	1 0	Cooling Water > 100 l/hr Cooling Water ≤ 100 l/hr

A warning signal (S = 1) will be generated if:
Either : (a) Temperature > 120C and Cooling Water <= 100 l/hr
Or (b) Temperature <= 120C and (Pressure > 10 bar or Cooling Water < 100 l/hr)

Draw a logic network and truth table to show all the possible situations when the warning signal could be received.

5. Draw a circuit diagram for δ = (xy' + x'y)z

6. Device a suitable Boolean expression and truth table for the circuit below:

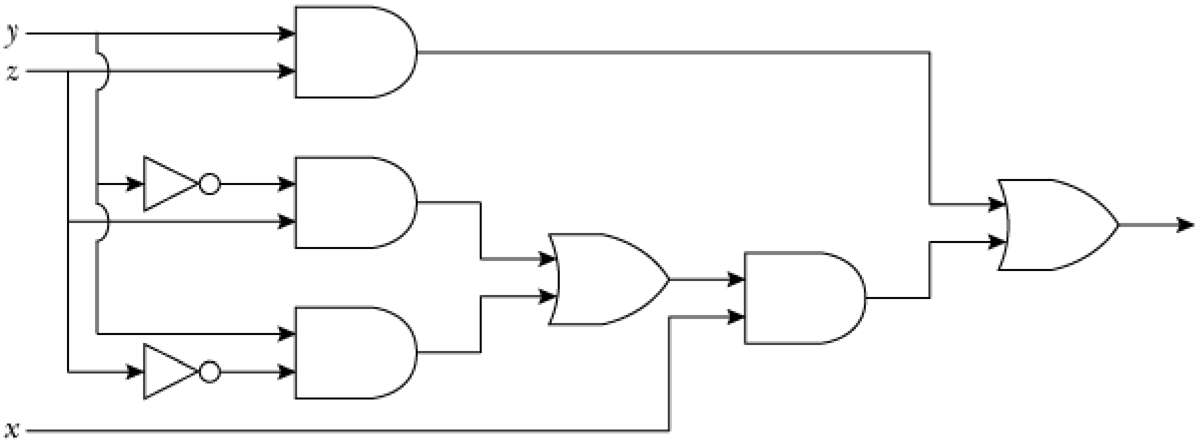


FIGURE 12.19 A circuit

7. Draw circuits for following Boolean statements.
a. If A AND B are on AND C AND D are on then output is on.
b. If A OR B are on AND C OR D are on then output is on.
c. If A OR B is on then output is off.
d. If B AND C is off OR A is on then output is on.
e. If A is off AND B OR C is on AND D is off then output is on.
f. If A is on AND B AND C are off AND D is on then output is on.
g. If smoke detector (S) is on OR fire alarm (F) is on then sprinkler (W) is on.

8. a. Simplify the following logic equations by using the rules of Boolean algebra.
a. A • C + A • B • C
b. (A + B) • (B + $\bar{A}$)
c. A • ($\bar{A}$ + C) + C
b. For each of the previous questions, create a circuit for the Boolean expression before simplification.

CHAPTER 9: DATA COMMUNICATION AND NETWORKING

Data Transmission/communication:

It is the process of transferring data through networked computers. It involves the transmission or passing of data and information from one computer (or device) to another.

For transmission to occur, there must be the following:

- Sender
- Transmission media
- Receiver

Transmitted data can be in **analogue** or in **digital** form.

- **Digital Data**: Data is in discrete value, that is, in ones and zeros. Digital data has the following advantages:
  - Digital data produces high quality output.
  - It is easier to represent.
  - Rebooting is easier
  - Data is compressed and therefore takes less disc storage space
  - In some

- **Analogue Data**: Data is in continuously varying form, e.g. human voice. This is difficult to handle as it will be in form of waves. Sensors collect data in analogue form, eg. 67, 93 are all analogue data

Transmission Media

Transmission media refers to the path through which data is transferred from one point to another. Transmission media can be either guided or unguided.

Guided Transmission Media

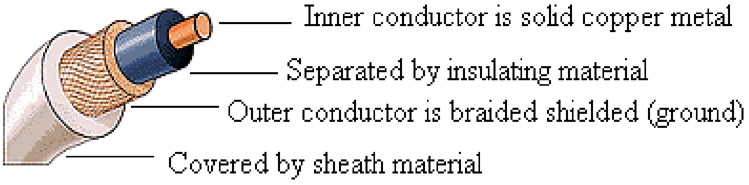
In guided transmission media, data follows a physical path during transmission, e.g. through a coaxial cable. The two connection points should be linked together by a physical communication pathway. Some of the guided communication media are:

- Unshielded Twisted Pair (UTP) cable
- Fibre optic cable
- Coaxial cable

1. Unshielded Twisted Pair: These are cables with two copper wires of about 1 millimetre thick. The wires are twisted to avoid crosstalk. Twisted pair is very cheap to buy and offer good performance over short distances.

Disadvantages of twisted pair: Twisted pair is very cheap to buy. Has big attenuation. Has low bandwidth

2. Coaxial cable: It is a stiff copper wire surrounded by an insulating material.

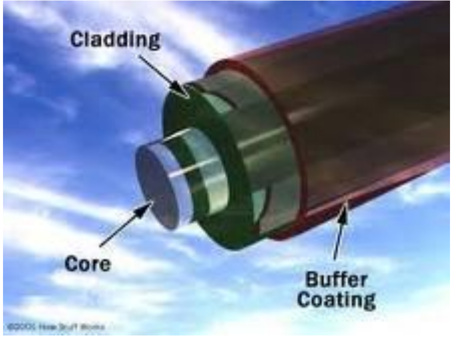

Inner conductor is solid copper metal
Separated by insulating material
Outer conductor is braided shielded (ground)
Covered by sheath material

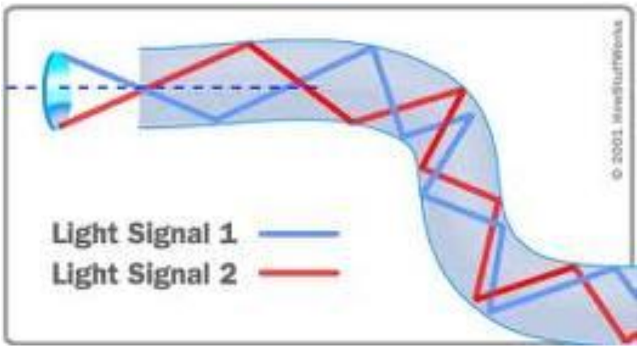
- Allows for both baseband and broadband transmission
- has less attenuation than twisted pair,
- has high bandwidth and
- Has low error rates.
- Can transmit analog and digital signals

- Ensures accurate data transfer.
- However, coaxial cable is expensive to buy and is stiff, making it difficult to handle.
- They are suitable for short distance communication on a LAN
- **Application**: Used for TV distribution (connecting decoders with the antenna on the satellite dish); long distance telephone transmission; short run computer system links, Local area networks

3. Fibre optic: A media that uses light to transmit data. Used in Wan and Man networks. Its benefits are:-

- It has less attenuation and therefore fewer repeaters are needed,
- has very high bandwidth and cannot corrode (not affected by corrosion),
- it is thin and therefore has less weight.
- It allows very fast data transfer,
- has no electromagnetic interference,
- is physically secure.





Fibre optics is in two forms, multimode and monomode. Multimode fibre optic cable carries 2 or more signals at a time, each at a slightly different reflection angle. This is used over short distances. Monomode (Single mode) cable carried one signal at a time and is appropriate for long distance communication.

However, fibre optics is very expensive to buy and is uni-directional (travels in one direction only). Cable cannot bend around tight corners. It is also difficult to interface with computers.

Unguided Transmission

Wireless Transmission media

1. **Bluetooth (***Refer to presentations)*
2. **Radio** *(refer to presentations)*
3. **WIFI (Wireless Fidelity)**
   It is a Wireless LAN(A local area network) that uses high frequency radio signals to transmit and receive data over distances of a few hundred feet; using Ethernet protocol. it is a set of standards that set forth the specifications for transmitting data over a wireless network. There must be a wireless router which enables wireless devices to connect to the network and to the internet.

- Range: Wi-Fi provides local network access for around a few hundred metres
- Speed: maximum of 54 Mbps,
- Provides local area network
- Limited to one subscriber
- Can be used where cables cannot run
- Wireless network adaptors are inbuilt withion most devices like laptops, therefore cheaper and easier to get.
- Tend to be slower if more devices are added to the network
-

4. **WIMAX (World Wide Inter-operability for Marking Access)**
   - a single WiMAX antenna is expected to have a range of up to 30 Kilometres
   - Speed: with speeds of 70 Mbps or more. As such, WiMAX can bring the underlying Internet connection needed to service local Wi-Fi networks
   - Can accommodate many subscribers
   -
5. **Infra-red waves:** Refers to data transmission in form of waves as through some remote controls of televisions. It has enormous bandwidth. However, infrared waves do not travel through obstacles like buildings, they only work for very short distances, affect eyes and consume a lot of power.
6. **Satellite transmission:** These include earth stations which communicate with geostationary satellites (36 000 to 80 000 km above the earth. These have high bandwidth and support very long distance communication. However, they have big attenuation and are slow in sending messages.

- This is a method of networking computers and computer devices without the use of cabling, e.g. using bluetooth, radio, satellite and infra-red.
- The devices that are used in wireless technology include:
  - 3G (Wireless Application Protocol (WAP)) mobile phones / cellphone / remote keypad/remote control/remote keyboard.
  - Infra-red mouse.
  - Multimedia mobile handsets and
  - Notebooks.
  - GPRS (general packet radio service) modems.

GPS - A navigational system involving satellites and computers that can determine the latitude and longitude of a receiver on earth by computing the time difference for signals from different satellites to reach the receiver

Advantages of wireless communication include:

- Cheaper as no wires are needed for networking.
- Ensures fast wireless Internet access, depending on the technology being used.
- Wireless LAN is faster than a modem or mobile.
- Wireless LAN enables working at home.
- Users can create and send multimedia messages to mobiles or e-mail while in transit.
- Users can send greetings from mobiles to mobiles/PCs.
- Ensures instant transmission.
- Users can download e-mail and file attachments while on mobile.
- Users can watch live web cast on mobile.
- Users can listen to streaming video on mobile.
- Users can watch news, weather, sport, games while on the move.
- Users can access information from mobile anytime.
- Users can send, receive, delete e-mail while on the move.
- Users can view business appointments while out of office on mobile.

- Users can send corporate e-mail while out of office - even behind a firewall on mobile.
- Users can use wireless internet connection from chat rooms for discussions with colleagues while on the move.

Disadvantages of Wireless Technology:

- Wireless LAN speeds are slower than Net access at work due to narrow bandwidth.
- Anyone within the Wireless LAN nodes range with an appropriate device can use your Wireless LAN and broad band link.
- Anyone who walks past your house or WLAN linked into a corporate system can access sensitive information like credit card details.
- 3G phones are not compatible with 2G phones.
- Blue tooth has limited range.
- Signals can be blocked, distorted or will be weak.
- Can lead to health problems from microwaves

Synchronous and asynchronous Transmission

Synchronous Transmission:

- This is whereby data is sent in blocks (packets) at any given time, and uses control characters.
- This method is faster in transmitting data.
- Data transfer is timed by the clock pulse
- There is no need for start and stop bits since the timing signals are used to synchronise transmission at sending and receiving ends.
- Mostly used in local area networks
- Many transmission errors are bound to occur.

Asynchronous Transmission:

- This is whereby data is send character by character over a transmission channel.
- This is much slower as compared to synchronous transmission.
- A **start** bit and **two stop** bits marks the beginning and ending of a character respectively.
- The start and the stop bit are always different.
- The start bit alerts the receiving end and synchronises its clock, ready to receive the character. The baud rate of the two devices is set to be similar so as to correctly receive the data.
- A parity bit is included to check against incorrect transmission.
- Each character is send as soon as it becomes available rather than waiting for the clock pulse

Serial data transmission

- Data is send one bit at a time over a single wire from source to destination, until all data has been send.
- A system in which one bit is send at a time along the same data line until the whole data has been send.
- Suitable for long distance communication since less cabling is needed.
- Very reliable form of transmission
- Can be fast using fibre optic cable

Parallel data transmission

- All the bits (byte) of a character are send simultaneously along separate data lines.
- Several bits are send simultaneously over a number of parallel lines linking the sender and the receiver.
- This is faster in data transmission
- Suitable for short distance communication, i.e, inside the computer system using buses, e.g. from processor to hard drive, processor to printer, etc.
- A parallel port is used to link external devices like printer to the processor.

- Bits can arrive at the destination at different times since the cables may have different speeds if the distance is too long. This is called skew. This is prevented by using short distance transmission.
- It becomes expensive in long distance communication system systems since too much cabling is required.
- It is more reliable over very short distances

Baseband Vs Broadband transmission

Baseband transmission

- This is a one-channel transmission system whereby the whole bandwidth is dedicated to one data channel.
- This is whereby a channel carries one signal at a time (either 1 or 0), meaning the presents or absence of voltage in the channel.
- Fast in transmission
- Suitable mostly for short distances
- For long distances, repeaters or boosters are required

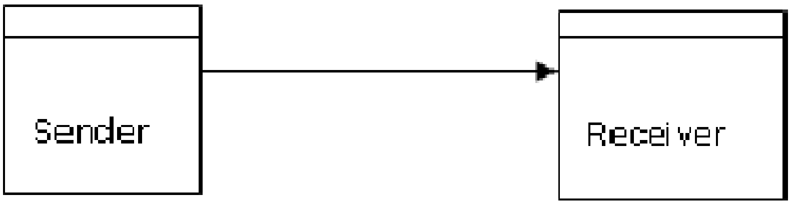
Broadband transmission

- This is a multi-channel system where several channels are combined into one carrier signal, where the bandwidth is shared by different channels.
- Broadband carries multiple signals on a fixed carrier wave.
- Bandwidth is shared by different channels.
- It offers faster transmission rates
- Enables transmission of voice, video, computer data, etc, simultaneously.
- Broadband is expensive to install and maintain
-

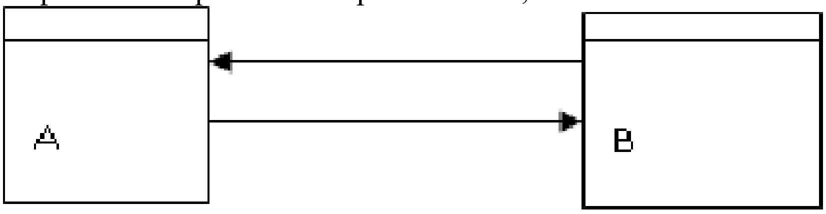
Transmission Modes

Transmission modes include Simplex, Duplex (Full duplex) and Half Duplex

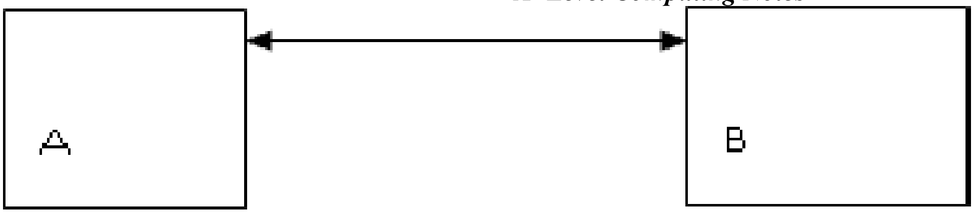
Simplex Mode: This is a mode of data transmission in which data travels only in one direction. Thus one computer acts as the sender and the other as a receiver at any given time. A good example is teletext service. See diagram below:



Half Duplex: This is a transmission mode in which data travels in both directions but not simultaneously. The receiver waits until the sender has finished sending data in order for him to respond. Examples include police radios, uses 'Over' to allows time for other to transmit



Duplex/Full Duplex: This is a transmission mode in which transmission is possible in both directions simultaneously. See diagram B above. There is no need for one to wait until the channel is free from data. Used by high speed mainframes on the internet.



Transmission impairments

This refers to change in signal form as it propagates through the transmission channel. Transmission impairments include:

Attenuation: The loss of signal power as it moves through the transmission channel.

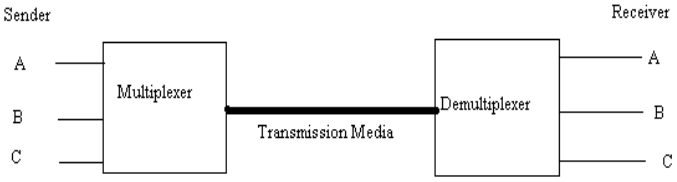
Noise: Occurs when an unwanted signal from other sources than the transmitter enters the transmission channel.

Distortion – means that the signals are deformed a more or less different signal as it propagates through the medium

Multiplexing

This is a method of allowing multiple signals to share the same channel, reducing too much cabling, as shown below:

A multiplexer is used in multiplexing. A multiplexer is a device that joins two or more channels into one channel while the de-multiplexer is responsible for splitting a channel into a number of them for easy transmission to the intended destination.



Bandwidth

Refers to the carrying capacity of a transmission channel. It is generally the volume of data that a communication channel can carry at a given time. It is the difference between the lowest and the highest (range) amount of data that a channel can transmit. It determines the amount of data a channel can transmit at a given period of time. Fibre optic cables have high bandwidth and therefore transmits data faster than coaxial cables, which have low bandwidth.

Baud rate: the amount of bits that can be send of a channel per second. It is a key measure of data transfer rate. One baud = one bit per second

Signal routing methods

Circuit switching:

- A signal routing method in which the path (route) is first established from sender to receiver immediately before transmission can start. An end-to-end path is established before communication can occur. Thus a circuit must be established first.
- The circuit is maintained for the whole duration of transmission. Thus provides a dedicated communication path between two stations and offers bandwidth that cannot be infringed upon by other users.
- Circuit switching has the following major phases:
  i. Circuit establishment – a station to station circuit established.
  ii. Data transfer – data is now sent over the dedicated channel.
  iii. Circuit disconnect – connection termination.
- There is no need for waiting period for data.
- Continuous transmission ensures better utilisation of the channel's bandwidth.

- Sender and receiver must send and receive data at the same rate
- The transmission path remains open (connected) until transmission is complete.
- After transmission, the path can now be released for others to use.
- If no path is established, transmission cannot occur
- Similar to normal telephone systems whereby a specific line is routed from point A to point B and is dedicated but not necessarily used all the time.
- Data is not necessarily split, thus is send as it is.
- Data signals are received in the same order they are send, therefore no need for processing at the receiving end.
- **Advantages Circuit switching (CS)**
  - no congestion.
  - dedicated transmission channel with guaranteed data rate.
  - More effective transmission
  - Less transmission errors
- **Disadvantages Circuit switching (CS)**
  - Channel reservation for duration of connection even if no data are being transferred is an inefficient media use process.
  - Long delays in call setup.
  - Designed for voice traffic (analog).

Packet switching:

- Data is first split into smaller chunks called packets (or datagrams) which may take different routes and then reassembles to the original order at their destination.
- Packets are routed to the next (intermediate) node along an appropriate route, which can store and transmit the packet until the destination.
- Each packet takes its own convenient path and then re-assembled at the receiving end.
- Packets do not necessarily arrive at the same time or in correct order.
- At the destination, packets are re-grouped to the original message.
- Packets can be of fixed size
- Each packet has the following data: source address, destination address, error control signal, packet size, packet sequence number, etc.

Benefits of packet switching:

- Makes more efficient use of lines
- Cheap as cost depends on number of packets send, not distance, so all data can be transmitted at local call rates
- Less likely to be affected by network failure since an alternative route is used from each node.
- Security is better since packets follow different routes
- No call set-up is required.
- Fast and suitable for interactive applications

NB: A virtual circuit must be established between the sender and the receiving end. Virtual circuit – A temporary 'dedicated' pathway between two communicating points on a Packet Switched System before sending of packets. Bandwidth is allocated for a specific transmission pathway.

Message Switching

- This is whereby the whole message may be routed by any convenient route.
- No physical/dedicated path is established in advance between sender and receiver
- Data is stored at a hop (which may be router) then forwarded one hop later.
- Each block is received in its entity form, inspected for errors
- Data is not transmitted in real time.
- Blocking cannot occur
- Delays are very common

- Sender and receiver need not be compatible since sending will be done by routers, which can change data format, bit rate and then revert it back to original format on receiving or submit it in different form.
- Storing data solves congested networks since data can be stored in queue and forwarded later when channel becomes free
- Priorities can be used to manage networks
- Very slow if the number of nodes is many since each node stores before forwarding the data
- In message switching, whole message is routed in its entirety, one hop at a time.
- Now implemented over packet or circuit switched data networks.
- Each message is treated as a separate entity.
- Each message contains addressing information, which is used by switch for transfer to the next destination.
- Also called a store and forward network
- Used in e-mails and in telex forwarding
- There is often no real limit on the message / block size.

Advantages

- more devices can share network bandwidth
- reduced traffic congestion
- one message can be sent to many destinations through broadcast addresses

Disadvantages

- often costly – must have large storage devices to hold potentially long messages
- not compatible with most real time applications

Transmission protocols

A protocol is a set of rules that govern how data is transferred in a network. It defines the rules on how network devices communicate, e.g the TCP/IP. This includes:

- how to interpret signals
- how to identify 'oneself' and other computers on a network
- how to initiate and end networked communications

A network communication protocol: a standard method for transmitting data from one computer to another across a network. Some of the protocols are:

i. **HTTP (HyperText Transfer Protocol)**
This is a protocol that defines the process of identifying, requesting and transferring multimedia web pages over the internet. It is used for transferring data across the internet, usually between servers and computers on the internet. It is based on the client –server relationship. It uses TCP/IP to transmit data and messages

ii. **FTP (File Transfer Protocol)**
it is a protocol used to transfer data from one computer to another. It is often used to download software from the internet, and it uses the TCP/IP protocol in doing this. However, FTP has no security to data as the data is not encrypted prior to its transmission.

iii. **TELNET**
This is a network protocol that allows a computer user to gain access to another computer and use its software and data, usually on a LAN and on the Internet. It allows users to access data stored on servers from their terminals. Telnet allows computers to connect to each other and allows sharing of data and files. Telnet has security problems especially on the internet.

iv. **VoIP (Voice Over Internet Protocol)**
It is a method of using the internet to make ordinary voice telephone calls. Thus it is a way of having phone conversations using the internet as a way of communication. By VoIP,

international and long distance calls are of the same price as local calls and sometimes are for free. However, the system does not offer emergency calls. An example of VoIP is Skype.

1. TCP/IP (Transmission Control Protocol/Internet Protocol)

- protocol governing the transmission of data
- data is divided into packets to which addressing information, error correction code and identification are added
- the packets travel to their destination over the network and the receiving PC checks for mistakes and pieces the data together in the right order
    - **TCP/IP (Transmission Control Protocol Internet Protocol)**

TCP: It ensures that data is transmitted accurately

IP: It ensures that data is transmitted to its correct address (IP address). Every device on the internet has its IP address. It also ensures that packets are rearranged to the original message on arrival of their destination.

2. OSI (Open Systems Interconnection)

This is a model of communication designed by the International Standards organization (ISO). The OSI model allows computers from different manufacturers or origin to be connected together. The idea is that suppliers must produce hardware or software to implement any of the seven layers while other suppliers provide those of other layers. This promotes standardization. The seven layers are:

- **Application Layer** – defines how user applications interface or access communication services, initiates or accepts a request, provide network applications like data transfer, messaging, operating system functions, etc
- **Presentation Layer** – deals in how information is presented, e.g ASCII, adds formatting or data transformation,(e.g. from ASCII to Unicode), data compression and data encryption.
- **Session Layer** – creates and terminates sessions, e.g. login session, file transfer session, adds traffic-flow control information, etc.
- **Transport Layer** - allows error correction during transmission, maintains flow control, allows data recovery, allows routing, addressing and multiplexing of signals.
- **Network Layer** - adds sequencing and address information, sets logical protocols, creating frames (consisting of address fields, control filed, date, and error control field)
- **Data Link Layer** - provides error-checking and formats data for physical transmission, type of network and packet sequencing is defined, used for syncronisation.
- **Physical Layer** – level of actual hardware. define physical characteristics of ntwork e.g connections, wiring, voltage signals, etc

At each level, additional information is added to allow service to be provided. This layered model is also called **protocol stack**

NETWORKING

Types of networks

i. LAN (Local Area Network

A LAN is a privately owned connection of computers on a very small geographical area for sharing of data and files by users of the network, for example, within an single room. Usually connected using cables of radio connections.

Hardware Requirements for a LAN

- Network Interface Card (NIC):- Each computer on the network must have this as it allows computers to be linked and to be uniquely identified on the network.
- Server:- to store software that controls the network, software and files and also data that can be shared by all users of the network

- Hub or alternatively a Switch:-

  A hub is a device that connects workstations together in order to make a LAN. It receives signal/data from workstations, regenerates it and the sends it to all ports on it. Thus all workstations connected to it will get the signal or data packets. Hubs are less intelligent, they do not determine the exact computer the data is addressed to and so they broadcast the signal. This is a security risk. It is usually used on a star network or on a hybrid network. A hub has many ports on which cables to all computers on the network are connected.

  A switch is a networking device that allows multiple devices and workstations to be connected to each other on a LAN just as a hub does. However, a switch is more intelligent than a hub. A switch directs traffic across a LAN, enabling computers to talk to each other and share resources. It joins computers on a LAN and is found at layer 2 of the OSI reference model. It allows different nodes on the network to directly communicate with each other. A switch runs in full duplex mode. It can recognise different devices on the network using their MAC address so that data and signals can be send to exact/intended devices. This is more secure than a hub. Switches can be LAN switches or ATM switches which are used on WANs and MANs.
- Terminals:- these are computers that are connected to each other through a server and cannot work without the server. Terminals can be dump or intelligent. A dump terminal does not have neither processing nor storage capabilities and thus wholly depends on the host computer for it to work. An intelligent terminal has limited processing and or storage capabilities.
- Workstation:- these are the computers connected to the server and are less powerful than the server
- Cables: - connects computers together and acts as pathway for data moving from one workstation to another.

- Bridge: - this is a device that connects networks using the same communication protocols. It is used to connect different parts of a LAN, thus is used to connect different LAN segments together. However, it cannot handle multiple paths for data. In general a bridge is used for:

  - ✓ Increasing the number of workstations on the network
  - ✓ Allows connection of different types of LANs
  - ✓ Allows different segments to be treated as one network
  - ✓ Allows easier management of the network
  - ✓ Improves network security

ii. WAN (Wide Area Network)

WAN refers to connection of computers over a very large geographical area and may cover the whole world. The internet is part of the WAN. A WAN is created by joining several LANs together, for example, connecting different branches of a company that are in different provinces or countries. Computers are usually linked together using fibre optic cables, satellite links, telephone lines, etc.

Gateways are usually used to bridge the different networks.

Hardware Requirements for a WAN

- Routers:- this is a network device that connect different types of networks together, for example, connects a school LAN to the internet (which is a WAN). It can route packets of the same protocol (e.g. TCP/IP) over networks with dissimilar architectures (e.g. Ethernet to token ring). It receives transmitted messages and forwards them to their correct destinations over the most efficient available route. A router is used to form complex networks with multiple paths between network segments (subnets), each subnet and each node on each subnet is assigned a network address.

A router is very intelligent. It uses network addresses and IP addresses of other routers to create routes between two networks. They keep tables of addresses that will be used in routing information. Routers are thus used for:

- ✓ Determining the path of data packets using destination addresses of the packets.
- ✓ Used for packet switching

- Gateway: - a device used to connect different kinds of networks. Thy act as link to different WANs. A gateway is a device that connects networks with different architectures and different protocols. When packets arrive at a gateway, the software strips all networking information from the packet, leaving only the raw data. The gateway translates the data into the new format and sends it on using the networking protocols of the destination system. Thus it becomes a protocol converter.

- Modem (MOdulator DEModulator):- This is a device that converts digital signal received from a computer into an analogue signal that can be sent along ordinary telephone lines, and back to digital at the receiving end. Mostly used to connect to the internet using the ordinary telephone line. The speed of modems is measured in bits per second e.g. 56K bps. The following parameters must be specified when a modem is installed:
  - ✓ the telephone number of the ISP
  - ✓ baud rate of modem
  - ✓ number of data bits per block
  - ✓ number of stop bits
  - ✓ whether odd or even parity is used

  Cable modems - employ broadband transmission across regular cable television wires

Communication terms used on WAN networks

- **Integrated Services Digital Network (ISDN) line** – it is a digital telephone service that provides fast, accurate data transmission over existing copper telephone wiring, for internet connection. It is a set of communication standards for simultaneous digital transmission of voice, video, data, and other network services over the traditional circuits of the public switched telephone network. ISDN is a line that allows the transmission of digital signals without them being changed into analogue which leads to improved quality for the user. It requires a network adapter and a network termination device (no modem required)
- Asymmetric Digital Subscriber Line (ASDL) - offers Internet connection up to 30 times faster than dial-up modems still using traditional copper wires but allocating more bandwidth to the data flow from the ISP to the PC than is allocated from the PC to the ISP
- Dial-up networking: user pays for the amount of time spent using the telephone link. It is less expensive. more appropriate for low-volume applications requiring only occasional transmission
- Dedicated/leased line: the line is continually available for transmission and the user pays a flat rate for total access to the line. It transmits data at higher speeds. It is more appropriate for high volume transmission
- Value-added network (VAN):- This is a private, multipath, data-only, third-party managed network. It is used by multiple organisations. It may use ISDN lines, satellite links etc. it is set up by a firm in charge of managing the network. Its subscribers pay a subscription fee and for data transmission time. The cost of using the network shared among many users. subscribers do not have to invest in network equipment or perform their own error checking, routing and protocol conversion

- Electronic data interchange (EDI):-e.g. transmitting A level results to schools using BT's CampusConnect . virtually instantaneous electronic transmission of business data from one firm's computerised information to that of another firm. It increases accuracy and eliminates delays.
- Internetwork:- This is created when two or more independent networks are connected but continue to function separately e.g. Internet. In larger networks it is common to supply multiple paths through the network to provide fault tolerance

iii. MAN (Metropolitan Area Network)

It is an extension of a LAN, but usually extends to a larger geographical area, usually the whole city and is owned by a consortium of users. Different LANs are joined together to provide service for the whole city or district. Organisations in the city have their LANs linked together so that local users can access their databases for services. Users can view their electricity bill, make payments and even receive broadcasts pertaining to issues within their area. The network is only accessed by a group of users as defined by the organisation, e.g students of a certain university. It enables users to do researches, share files, libraries, local email and video conferencing.

NETWORK TOPOLOGY (CONFIGURATION)

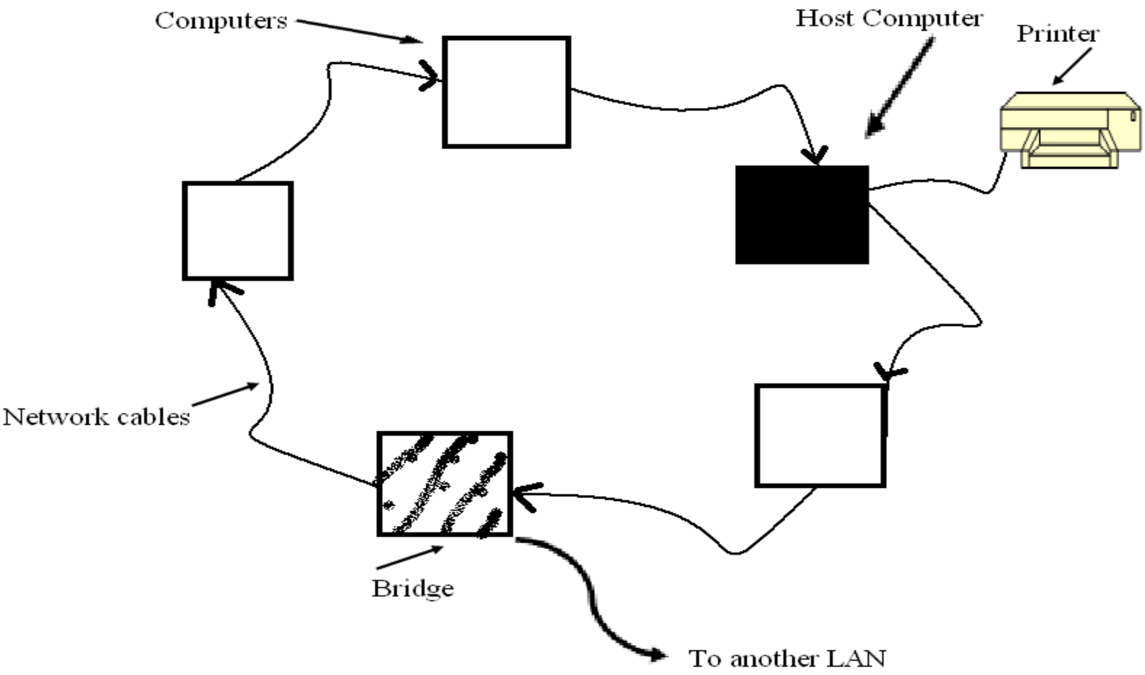
This refers to the shape/configuration of the network. This may refer to logical or physical configuration. The shape of the cabling layout used to link devices is called the physical topology of the network. This refers to the layout of cabling, the locations of nodes, and the interconnections between the nodes and the cabling. The physical topology of a network is determined by the capabilities of the network access devices and media, the level of control or fault tolerance desired, and the cost associated with cabling or telecommunications circuits.

The logical topology is the way that the signals act on the network media, or the way that the data passes through the network from one device to the next without regard to the physical interconnection of the devices. Logical topology is not necessarily the same as its physical topology. For example, the original twisted pair Ethernet using repeater hubs was a logical bus topology with a physical star topology layout. Token Ring is a logical ring topology, but is wired a physical star from the Media Access Unit.

Common network topologies are Ring, bus, star, mesh and hybrid network topologies.

1. Ring (Token ring) Network:

- Computers are connected together to form a circle and uses a token when transferring data.
- It uses token ensure that there is no collision
- Only the computer with a token can transmit.
- When a computer wants to send data, it sends it together with the token. Every computer on the network examines that data, if it is addressed to it, it copies/removes it and passes it on, until it comes back to the sender.
- Data/information travels in one direction only.
- Information moves around the ring in sequence from its source to its destination.
- As data passes from one computer to another in the ring, each computer removes any data relevant to itself and adds any data it wishes to send.
- The diagram below illustrates the physical configuration of a ring network:



Token passing

- a small packet called a token is passed around the ring to each computer in turn
- to send information, a computer modifies the token, adds address information and sends it down the ring
- information travels around the ring until it reaches its destination or returns to the sender
- when a packet is received by the destination computer, it returns a message to the sender indicating its arrival

<u>Advantages of Ring Network</u>

- Data processing is faster as each computer processes its own processor.
- Has very high data transfer rates.
- Uses a token to avoid data collision or loss
- it is possible to create large networks using this topology
- If one computer breaks down, others will remain working as they have their own processors and storage facilities.
- Performs better than star network when traffic is very heavy.

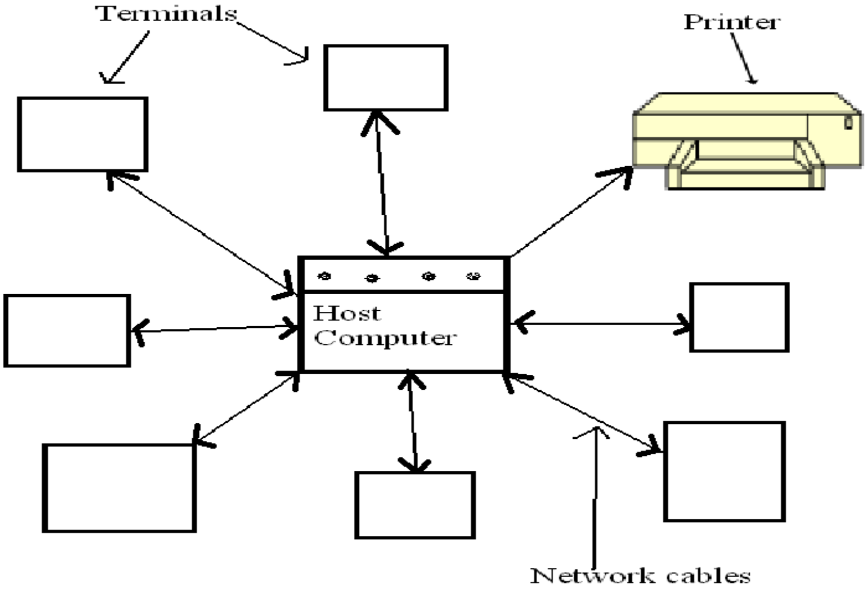
<u>Disadvantages of Ring Network</u>

- If one computer breaks down, the whole network is disrupted.
- a faulty connection between two stations can cause network failure
- Its requirements are expensive, that is buying several computers with processors and storage facilities.
- It is difficult to link the computers together.
- Difficult to add another computer without disrupting the networking.
- Only the computer with the token is allowed to send data at a given time. One may not send data when another node (computer) is still sending its own data.
- System is less secure as token together with data has to pass through other nodes that do not concern it.

<u>2. Star Network:</u>

Computers form a star shape with host computer at the centre.

The Server (host computer) manages all other computers/terminals on the network.
If the terminals are not intelligent, they have to rely on the host computer for everything.
This network is as shown below:



- computers connected by cable segments to a central hub/switch
- a signal sent from a computer is received by the hub and retransmitted down every other cable segment to all other computers on the network
- only the computer the signal is addressed to acts upon the data
- if one computer fails, the others are unaffected
- if the hub goes down, the whole network goes down

Advantages of Star Network

- If one terminal breaks down, the network is not disrupted.
- It is cheap in terms of requirements since only the host computer can have a processor and storage facility.
- It is very easy to install.
- Management of data is easier as this is done centrally.
- It is very fast to process data.
- Easier to add new workstation without disrupting the network.
- No problem of collision of data since each terminal has its own cable to the host computer.
- Gives consistent performance even when the network is heavily utilised.
- More secure than other network topologies
- it is easier to identify faults using this type of topology
- It is easy to expand this type of network
- If one terminal breaks down, others will remain working.

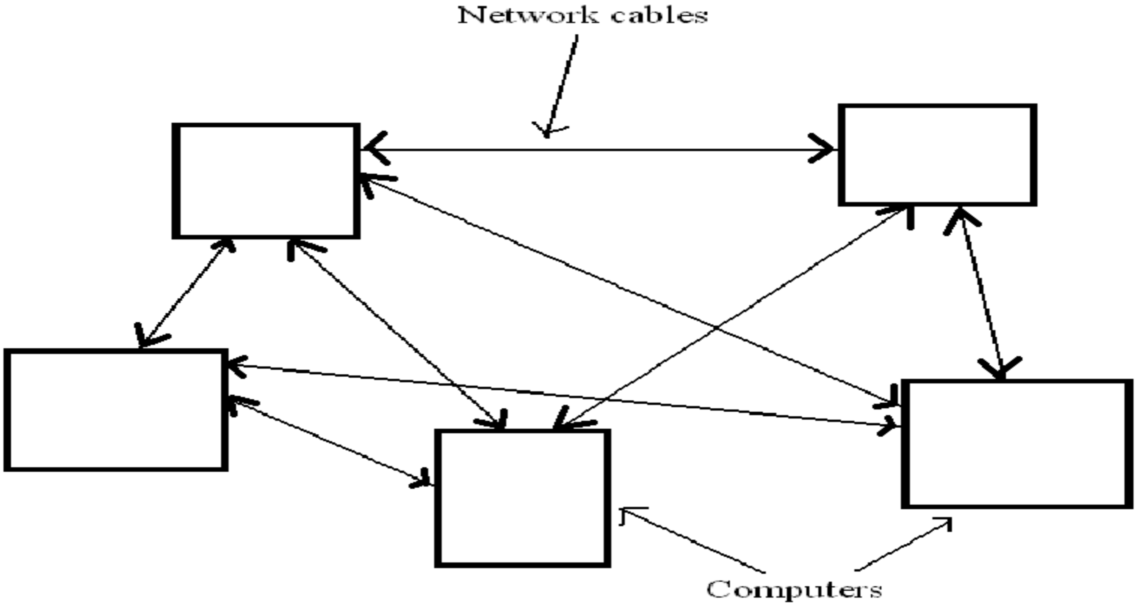
Disadvantages of a Star Network

- If the host computer breaks down, the whole network will be disrupted.
- If the host computer is down, all the terminals will not work as they depend on the host for processing and storage.
- It requires a lot of cabling, which might be expensive.
- Can be slower if overloaded

<u>3. Mesh Network</u>

- A network in which each computer serves as a relay point for directly sending information to any other computer on the network.

- No central device oversees a mesh network, and no set route is used to pass data back and forth between computers.
- Thus, if any one computer is damaged or temporarily unavailable, information is dynamically rerouted to other computers—a process known as *self-healing*



Advantages of Mesh Network

- If one computer breaks down, others will remain functional.
- If one computer breaks down, the network is not disturbed.
- Computers have their own storage and processing capabilities.

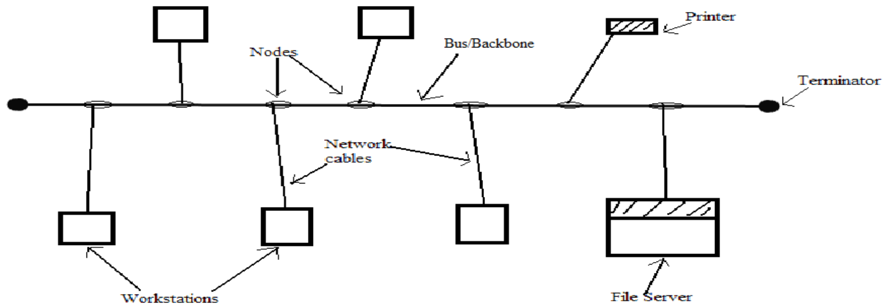
Disadvantages of Mesh Network

- Expensive to buy computers with their storage and processing facilities.
- Too much cabling is involved, which may be expensive.

4. Bus (Linear/Multi-drop) Network:

Computers are connected together through one main cable and all computers all signals transmitted by any computer connected to the network. Computers use CSMA/CD when sending data.

- all components are connected via a backbone (a single cable segment connecting all the computers in a line)
- entire network will be brought down by a single cable break
- terminator at the end of the line absorbs all signals that reach it to clear the network for new communication
- data is sent in packets across the network and received by all connected computers; only the computer with the packet destination address accepts the data
- only one computer can send information at a time
- Ethernet uses a collision system - carrier sense multiple access with collision detection (CSMA-CD) - if transmitted messages collide, both stations abort and wait a random time period before trying again.
- network performance degrades under heavy load



Definition of Terms

(a) **Bus/Backbone**: the dedicated and main cable that connects all workstations and other computer devices like printers.

(b) **Nodes**: these are connection points for workstations and the bus.

(c) **Terminator**: devices that prevent data in the bus from bouncing back, causing noise and prevents data from getting lost.

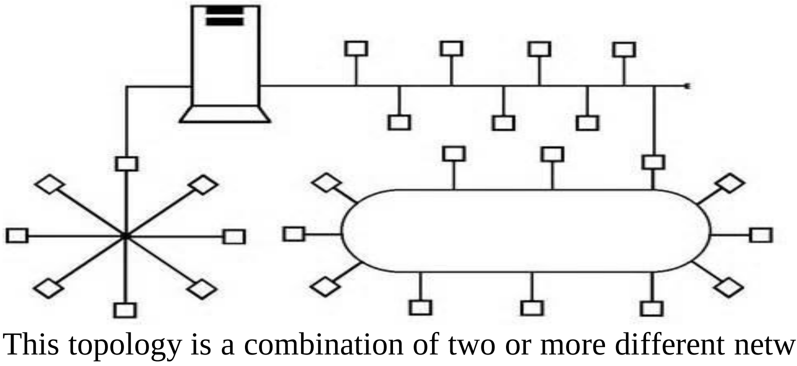
Advantages of Bus network

- If one workstation breaks down, others will remain functional.
- If one workstation breaks down, the network remains working.
- All computers have processing and storage capabilities.
- It is cheap to install due to less cabling.
- Easy to add workstation without disrupting the network.
- Requires less cabling than a *star* network.
- Less expensive network than the other systems

Disadvantages of Bus Network

- Computers cannot send data at the same time nor while there is data being transferred in the bus.
- Can cause collision of data during transmission.
- It is slow in transferring data.
- Its requirements are expensive, that is computers with their own processors and storage facilities.
- The system will be down if the main cable (bus) is disrupted at any point.
- Less secure.
- Performance worsens as new stations added

5. Hybrid



This topology is a combination of two or more different network topologies into one. When different topologies are connected to one another, they do not display characteristics of any one specific topology.

Advantages of hybrid topology

1. **Flexibility**:- adding / removing other peripheral connections is easy.
2. **More reliable**: it is easier to isolate the different topologies connected to each other and find the fault with the hybrid topology.
3. **Speed**: Speed is consistent, combines strengths of each topology and eliminates weaknesses

4. **Effective:** The weaknesses of the different topologies connected are neglected and only the strengths are taken into consideration.
5. **Scalable:** It is easy to increase the size of network by adding new components, without disturbing existing architecture.

Weaknesses of hybrid topology

1. Since different topologies come together in a hybrid topology, managing the topology becomes difficult.
2. It is also very expensive to maintain. The cost of this topology is higher as compared to the other topologies. Cost factor can be attributed to the cost of the hub/switch, which is higher, as it has to continue to work in the network even when any one of the nodes goes down.
3. **Costly Infrastructure:** The cost of cabling also increases, as a lot of cabling has to be carried out in this topology.
4. Installation and configuration of the topology is difficult since there are different topologies, which have to be connected to one another.

NB: Point – to-Point Connection: Point-to-point topology is the simplest connection, consisting of two connected computers.

Media Access Methods

A. **Carrier Sense Multiple Access (CSMA): CSMA is a** contention access method in which each station first listens to the line before transmitting data. It first of all checks if there is data in the transmission channel before transmitting. Thus it cannot transmit while another device is transmitting.

a) CSMA/CD (Carrier Sense Multiple Access with Collision Detection)

This is an access method in which a station transmits whenever the transmission medium is available and retransmits when collision occurs. A device first listens before transmitting, and if the channel is idle, it sends the data. If the channel is busy, it continues listening until the channel is no longer busy. However, two device (stations) may be listening at the same time and then transmit simultaneously when they detect that the channel is idle. This causes collision. The transmitting devices detects that collision has occurred, and they cancel all the data in transmission, broadcast a message to other channels that collision has occurred. These channels are then given a random period of time to start listening again in-order to re-transmit.

CSMA/CD control software is relatively simple and produces little overhead. CSMA/CD network works best on a bus topology with burst transmission

Disadvantages

- CSMA/CD protocols are probabilistic and depends on the network (cable) loading.
- Considered unsuitable for channels controlling automated equipment that must have certain control over channel access. (This could be OK for different channel access).
- We can set priorities to give faster access to some devices (This is, probably, not an issue in some applications)

b) CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance)

It is an access method in which collision is avoided.

Refers to Class Presentation by Nicole- for CSMA-CD/CSMA-CA, Token passing and Contention

c) Token passing

- In token-passing systems, a small frame (the token) is passed in an orderly fashion from one device to another.

- A token is a special authorising message that temporarily gives control of the channel to the device holding the token.
- Passing the token around distributes access control among the channel's devices.
- Each device knows from which device it receives the token and to which device it passes the token.(see fig.)
- Each device periodically gets control of the token, performs its duties, and then retransmits the token for the next device to use.
- System rules limit how long each device can control the token.
- Whenever the network is unoccupied, it circulates a simple three-byte token.
- This token is passed from NIC to NIC in sequence until it encounters a station with data to send.
- That station waits for the token to enter its network board. If the token is free the station may send a data frame.
- This data frame proceeds around the ring regenerated by each station.
- Each intermediate station examines the destination address, if the frame is addressed to another station, the station relays it to its neighbor.
- If the station recognizes its own address, copies the message, checks for errors, and changes four bits in the last byte of the frame to indicate address recognized and frame copied.
- The full packet then continues around the ring until it returns to the station that sent it.

Advantages

- Even though there is more overhead using tokens than using CSMA/CD, performance differences are not noticeable with light traffic and are considerably better with heavy loads because CSMA/CD will spend a lot of time resolving collisions.
- A deterministic access method such as Token Ring guarantees that every node will get access to the network within a given length of time. In probabilistic access method (such as CSMA/CD) nodes have to check for network activity when they want to access the network.

Disadvantages

- Components are more expensive than for Ethernet or ARCnet.
- Token Ring architecture is not very easy to extend to wide-area networks (WANs).
- Token Ring network is much more expensive than Ethernet. This is due to the complex token passing protocol.

d) Contention

- With contention systems, network devices may transmit whenever they want.
- No referee mandates when a device may or may not use the channel.
- This scheme is simple to design
- The scheme provides equal access rights to all stations.
- Stations simply transmit whenever they are ready, without considering what other stations are doing.
- Unfortunately, the "transmit whenever ready" strategy has one important shortcoming.
- Stations can transmit at the same time.
- When this happens, the resulting co-mingling of signals usually damages both to the point that a frame's information is lost.
- This unhappy event is called a "collision."

Polling Access Method

- Polling is an access method that designates one device (called a "controller", "primary", or "master") as a channel access administrator.

- This device (Master) queries each of the other devices (“secondaries”) in some predetermined order to see whether they have information to transmit.
- If so, they transmit (usually through the master).
- Secondaries may be linked to the master in many different configurations.
- One of the most common polling topologies is a star, where the points of the star are secondaries and the master is the hub.
- To get data from a secondary, the master addresses a request for data to the secondary, and then receives the data from the secondary sends (if secondary sends any).
- The primary then polls another secondary and receives the data from the secondary, and so forth.
- System limits how long each secondary can transmit on each poll.

Advantages

- Polling centralizes channel access control.
- Maximum and minimum access times and data rates on the channel are predictable and fixed.
- Priorities can be assigned to ensure faster access from some secondaries.
- Polling is deterministic and is considered suitable for channels controlling some kinds of automated equipment.

Disadvantages

- Polling systems often use a lot of bandwidth sending notices and acknowledgments or listening for messages.
- Line turnaround time on a half- duplex line further increases time overhead.
- This overhead reduces both the channel's data rate under low loads and its throughput.

CHAPTER 10: FILE HANDLING

Types of files

1. Master File: it is a permanent file kept up-to-date by applying transactions that occur during the operation of the business. It contains all information of the job. It contains permanent and semi-permanent data. Static (permanent) data stored in database files can include Surname, First names, Date of birth, etc.

2. Transaction Files: These are temporary files that contain data that can change regularly, usually created on daily basis and is used to update the master file. It contains details of all transactions that have occurred in the last period only. This includes sales per day, student mark in a weekly test, etc. Transaction files are used to update master files. They can be discarded once updating has occurred.

3. Reference files: These are files that contain permanent data which is required for reference purposes only. This includes data on tax bands, formulae, etc. No changes to files are done.

4. Data file: A set of related records (either written or electronic) kept together.

5. Text file: file that only accommodated string data, no graphs, pictures or tables. Characters can be organised in a line by line basis.

FIXED LENGTH RECORDS

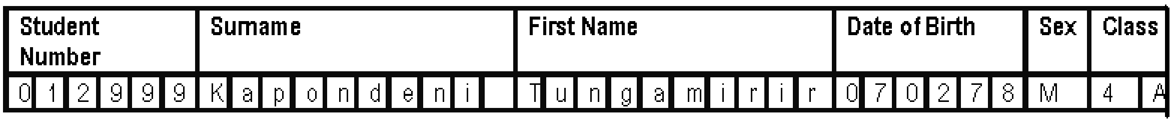
These are records that allocate a specific amount of space for data, mostly a specific number of characters. Every field contains exactly the same number of bytes. Also, every record contains exactly the same number of fields. For instance, a school keeps student records in a fixed length file. The student number has 6 characters, Surname was assigned 10 characters, First Name is given 10 characters, Date of Birth has 6 characters, sex has one character and class has 2 characters only in that order in the computer database file. In total, the length of each record is 35 characters.

The following student details are to be entered into the computer:

Student Number: *012999,* **Surname**: *Kapondeni,* **First Name**: *Tungamirirai*

Date of Birth: *7th of February 1978,* **Sex:** *Male,* **Class**: *Form 4A*

When entered into the database, the record will appear as follows:



Student Number	Surname	First Name	Date of Birth	Sex	Class
012999	Kapondeni	Tungamirir	070278	M	4A

From the table above, it can be noticed that:

- The Sex field is coded to accommodate only the letters *M* or *F*. This is shorter and therefore faster to enter data into the computer and to search records than entering the words Male or Female.
- The Surname *Kapondeni* is shorter than the allocated 10 spaces. The other spaces will remain idle.
- The First Name *Tungamirirai* is too long than the allocated spaces and therefore extra characters will be cut.

Fixed length records have the following advantages:

- Entering data is faster as records are shorter and less typing is required.
- Less memory is required.
- Less data entry errors are encountered.
- It is faster to carry out searches.
- Faster to do validation checks and procedures.
- They are easier for programmers to work with than variable length records.
- They allow an accurate estimate of disk storage requirements. Thus disk storage space can be easily managed as records occupy a specific number of characters.
- They are very easy to update.
- Faster to access records.

However, fixed length records have the following disadvantages:

- Can lead to wastage if disk storage space if used to store variable length data.
- For example, not all surnames are of the same length.
- Some spaces may lie idle as data entered will be shorter than the space allocated.
- Some data to be entered may be too long for the space allocated and therefore will be cut.

b. VARIABLE LENGTH RECORDS

These are records that allow data to occupy the amount of space that it needs. They allow data with varying (different) number of characters or sizes. Records may also have varying number of fields. The number of bytes in a particular field may vary from record to record. Also, the number of fields may vary from record to record. They usually show where the field or record starts and ends, for example:

0	1	2	9	9	9	*	K	a	p	o	n	d	e	n	i	*	T	u	n	g	a	m	i	r	i	r	a	i	*	0	7	0	2	7	8	*	M	*	4	A	˜

NB:- ** Indicates the end of field marker, and the ≈ indicates the end of record marker, and these* ***allow data to be processed***.

Variable length records have the following advantages:

- They are more economical in terms of usage of disk storage space as they do not allow spaces to lie idle.
- Less space is wasted on the storage medium.
- It allows as many fields as possible to be held on a particular record, e.g subjects taken in an exam by a particular student.
- More records can be packed on one physical block, thereby reducing time spend in reading the file.
- Data entered will not be cut but appears as entered no matter how long it is.
- No truncation of data occurs

However, variable length records have the following disadvantages:

- End of field and end of record markers occupy disk storage space that might be used to store data.
- These records are difficult to update as the transaction and master files might have different lengths.
- The processing required to separate out the fields is complex.
- It is difficult to estimate file sizes accurately when a new system is being designed.
- Records cannot be updated insitu.

FILE ORGANISATION

Refers to the way in which records in a file are stored, retrieved and updated. This affects the number of records stored, access speed and updating speed. The most common methods of file organisation are: Serial **File Organisation, Sequential File organisation, indexed – sequential file organisation and random (direct**) file organisation.

1. Serial File Organisation: This is whereby records are stored one after another as they occur, without any definite order as on magnetic tapes. Data is **not** stored in any particular sequence. Data is **read from** the first record until the needed data is found. **New records** are **added** to the end of the file. Serial file organisation is not appropriate for master files since records are not sorted and therefore are difficult to access and to update. Suitable for temporary/ transaction files since records are not sorted.

To delete records:

- More complex
- Read record to be deleted from the file, search it from 1st record until found=true
- re-write the whole file to a new disk, **omitting** the unwanted record.

To add a new record(algorithm);

- open the file
- append new record to the end of the file

2. Sequential File Organisation: This is whereby records are stored one after another and are sorted into a key sequence, that is, in ascending or descending order of a given key filed as on magnetic tapes. Records are held one after another in key sequence. Sequential files organisation is appropriate for files with a high **hit rate** like payroll processing.
They are suitable for master files since they are ordered. However, it takes too long to access required data since the records are accessed by reading from the first record until the required data is found. Adding of new records is difficult as this is done by re-entering the data and the new record is inserted at its right position. It is time consuming to update such records. **Suitable for master files** since records are sorted. This is used where **all** records need processing. They are faster and more efficient than serial files.
To **access/view a record**, each record on the file must be read, starting from the beginning of the file, until the required record is found.

To **add a new record**, copy existing records up to where the new record is to be inserted, insert record, then copy rest of file. The algorithm can be as follows:

- open old master file for reading
- open new master file for writing
- start from beginning of old master file
- Repeat
    - ✓ Read next record (call it current record)
    - ✓ If current record key>new record key THEN
        - ▪ Write new record to the new file
    - ✓ End If
- Until new record is inserted or EOF (old)
- If record not yet inserted THEN
        - ▪ Write new record to the new file
- Enf If
-

To **delete a record**, the whole file is to be copied over to a new sequential file, omitting the file to be deleted.
Processing of records is faster than that of serial files

<u>Hit rate</u> – proportion or percentage of records being accessed on any one run. In payroll systems, the hit rate is mostly 100% since every employee will be paid. Hit rate is calculated by dividing the number of records accessed by total number of records in the file and then multiplying by 100. For example, if 270 records are accessed out of 300 records, the hit rate is $^{270}/_{300}$ x 100 = 90%

3. Indexed-Sequential Files: This is whereby records are ordered in sequence based on the value of the index or disk address as supported by hard disks. It supports **batch processing.** It is also used for creating master file since the records are ordered. It is also suitable for real time processing applications like stock control as it is fast in accessing records and in updating them. It provides direct access to data as on hard disks, diskettes and compact disks. It ensures that data is accessed in some order. It ensures that no data is missed during accessing. Can provide direct access if requests are send online.
Indexed sequential files consists of 3 basic parts:

- ✓ the index
- ✓ The home area

- ✓ Overflow area

The index:

Contains record keys and disk addresses. The record key can be one or more fields that uniquely identify a record. Each record key is associated with a disk address (which can be surface, track and sector number) to identify the specific sector of the home area. Thus the index points to the home area.

The Home Area

This contains the data records stored in record key sequence. The home area is in sequence and can be accessed sequentially. In some situations, it can be accessed randomly using the index.
It allows data to be stored in blocks that contain several records. A block may be one or more sectors of the disk.
Each block may be partially filled in order to allow new records to be added later. For example, if a block can accommodate 12 records, 8 records may be saved in each block, allowing new records to be added during execution. This is called packing density, which is usually 70% or more. Thus if the computer is using 70% packing density, it means, data is stored in 70% of each block in the home area. The packing density is always less than 100% to allow insertion of additional records later.
The home area also points to the overflow area.

Overflow area

The home area may become too small and may not accommodate all records. The home area may become full. In this case, the remaining part of the home area just store pointers to indicate position of overflow area of any additional records as the home area gets full

NB: However, it may take longer to process the records. This is because records would have been placed in the overflow area. After reading the index, it takes a single disc access to read a record in the home area. Each time the home area is accessed, it takes at least two disc accesses; one to read the home area and one to read the overflow. This problem can be solved by re-organising the file using the housekeeping program, which copies the file to a new file, placing all the overflow records into the home area and re-writing the indices.

4. Random (Direct/Hash/Relative) File Organisation: This is whereby records are not in any order but stored and accessed according to their disk address or relative position, calculated from the primary key of the record, as supported by hard disks and compact disks. Records are stored and retrieved according to their disk address / relative position within file. The hashing algorithm/formula translates the primary key into an address, using the modulo method.
To **add a new record**, use the hashing algorithm to work out the appropriate memory location. If the location is empty, the records is inserted/written, otherwise the next block is examined until an empty space is found.
To **search/access** a record, its address is calculated from the record key using the hashing algorithm, the record at that address is then read, if it not the required record, the next record is read and examined until either the record is found or empty space is encountered. Suitable for online systems where fast response is required.
To **delete a record**, set flag to zero but leave the value there, therefore space can be reused but is not actually empty. The record is not physically deleted but just marked as deleted.

Structure of random files

Records are stored in blocks which are not necessarily in sequence. The position of the record is determined by a **hashing algorithm** or **randomising function**. When a record is to be stored in a file, a hashing algorithm is applied to the record key to determine the block that is to be used. For example, the blocks may range from 0 to 499, and the hash algorithm generates the number within this range. The records are stored in this format:

Record Key	Block

22387	300
13495	201
58905	104
48676	349
68798	34

It is appropriate where extremely fast access to data is required as in airline reservation. **Updating of records is in situ**, very simple and very fast. Hard disk, compact disks and diskettes promotes random file organisation.
When records are deleted, they are just marked as deleted but are not removed from the file. These deleted files take up space and may slow down processing. This can be solved by saving the records on a different file, removing the deleted records.

Overflow
If there is no space on the block, collision is said to have occurred and the record must be stored elsewhere.
A re-hashing algorithm is carried out on the block that is full in order to give another block that is not full. If the given block is full again, the hashing algorithm is applied again until an empty block is found. The overflow area can be used just as in the indexed sequential files.
NB:- If no further information is given, assume that overflow records are stored in the next block

Hashing algorithm - used to translate record key into an address. However, synonyms may occur, i.e. two record keys generate the same address (use overflow area and flag)

To solve problems of clashes of blocks after applying the hashing algorithm:

(a) Subsequent locations are read until empty location is found. The record is inserted in the empty location. If the maximum address is reached, it loops back to the first address, i.e. position 000
(b) A bucket (area of memory) can be set aside for overflow. Any clashing record is inserted in the bucket or in next location in serial form.
(c) Another method is to use the existing record as the head of list. Pointers are then used to point to records with the same hash value. New values are inserted in free location.

FILE PROCESSING

Refers to any form of activity that can be done using files. This includes: file referencing, sorting, maintenance and updating.
1. File Referencing/Interrogation: This involves searching of record and displaying it on the screen in order to gain certain information, leaving it unchanged. The record can also be printed.
2. Sorting: Refers to a process of arranging (organising) records in a specific ordered sequence, like in ascending or descending order of the key field.
3. Merging Files: This is the process of combining two or more files/records of the same structure into one. Below is an example of how records can be merged:

Record A (sorted)

12	34	71	78	101

Record B (unsorted)

103	67	3	90	12

Record C (Merged and sorted for records A and Record B)

3	12	34	67	71	78	90	101	103

4. File maintenance: This is the process of reorganising the structure of records and changing (adding or removing or editing) fields. May also involve updating more permanent fields on each record, adding / deleting records. This can be due to changes due to addition or deletion of records.
5. File Updating: Updating is the process of making necessary changes to files and records, entering recent information. Only master files are updated and they must be up-to-date**. For updating to occur, any one of the following must have occurred:**

A new record has been entered. Deletion of an unwanted record. An amendment (change) to the existing data has been made, e.g. change in date of birth only.

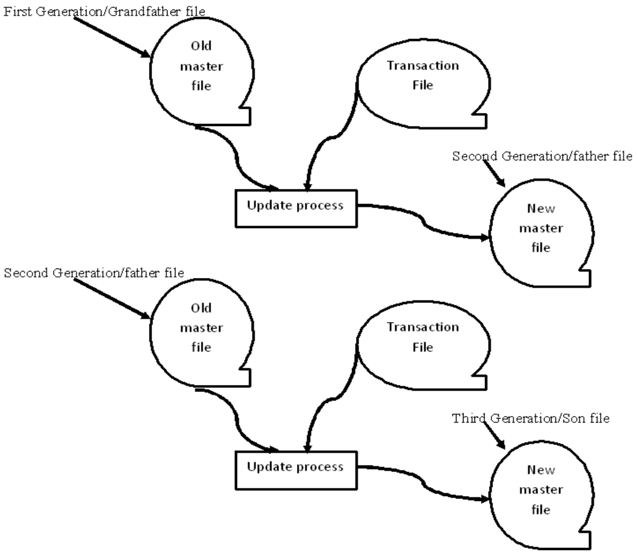
The most common methods of file updating are:
Updating **in situ** and Updating **by copying**.

a. Updating by copying

This happens in sequential file updating. The transaction file must be sorted in the same order with the master file records. This is done through the following steps:

- A record is read from master file into memory.
- A record is then read from transaction file into memory.
- Record keys from each file are compared.
- If record keys are the same, the master file is updated by moving fields form transaction file to the new master file.

In sequential file updating, it is recommended to keep at least three master file versions that will be used for data recovery in case of a system failure or accidental loss of data. The first master file is called the Grandfather file, the second master file is called the father file and the third master file is the son file. This relationship is called the grandfather-father-son version of files. The process of keeping three versions of master files (grandfather-father-son) as a result of sequential file updating is called **File Generations**. Thus the first master file (grandfather file) is called the first generation file, the second master file (father file) is called the second generation file and the third master file (son file) is the third generation file. The following diagram illustrates the sequential file updating process:



***NB: -** Always create data backups on compact disk or hard disks and re-run the old master file with the transaction file if the computer system fails or if data is lost. This is a data recovery method that works well.

***NB**:- A backup is a copy of file(s) on an alternative medium like CD-ROM in case the original file is damaged or lost and will be used for recovery purposes. The original files could be deleted accidentally, deleted by hackers, corrupted by system failure or could be corrupted by hackers.

Algorithm for sequential file updating

```
Open master file for reading
Open transaction file for reading
Open new master file for writing
Repeat
Read next transaction file record
While master file record key<transaction file record key
    Write master file record key to new master file record
    Read next master file record
End While
```

```
Update record
Until EOF (Transaction file)

While not EOF Master File
        Read next record
        Write master record to new master file
EndWhile
```

b. Updating by overlay (in situ):

In this case, record is accessed directly, read into memory, updated and written back to its original position (in situ). This occurs in random and indexed-sequential files, thus on devices like hard discs and memory sticks.

It applies for random files since record is accessed by means of an address therefore can be written back to same address after updating process.