**Cmp 111 INTRODUCTION TO COMPUTER**

Course Objective

To be able to define *computer program*, *algorithm*, and *high-level programming language*.

To be able to list the basic stages involved in writing a computer program.

To be able to list the major components of a computer and describe how they work together.

To be able to distinguish between hardware and software.

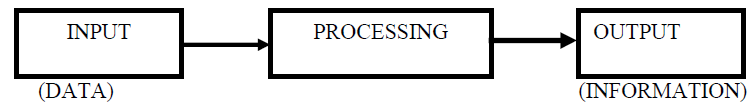
To be able to apply an appropriate problem-solving method for developing an algorithmic solution to a

problem.

History of computers, functional components of computer, characteristics of a computer, problem solving, flow charts, Algorithms, computer programming, Statements, symbolic names, Arrays, subscripts, expressions and control statements. Introduction to BASIC OR FORTRAN programming language, computer applications.

**Definition of a Computer**

A computer is an electronic device, use for manipulating data according to a list of instructions. A computer is a machine that is used to process data. In a more formal definition, a computer is any machine or device which, under the control of a stored program, can accept data in a prescribed form, process the data, and supply the results as information in a specified form.



**Schematic diagram to define a computer**

**Data:** The term data is referred to facts about a person, object or place e.g. name, age, complexion, school, class, height etc.

**Information:** Is referred to as processed data or a meaningful statement e.g. Net pay of workers, examination results of students, list of successful candidates in an examination or interview etc.

**Methods of Data Processing**

The following are the three major methods that have been widely used for data processing over the years:

a. Manual method

b. Mechanical method and

c. Computer method.

**Manual Method**

The manual method of data processing involves the use of chalk, wall, pen pencil and the like. These devices, machine or tools facilitate human efforts in recording, classifying, manipulating, sorting and presenting data or information. The manual data processing operations entail considerable manual efforts. Thus, manual method is cumbersome, tiresome, boring, frustrating and time consuming. Furthermore, the processing of data by the manual method is likely to be affected by human errors. When there are errors, then the reliability, accuracy, neatness, tidiness, and validity of the data would be in doubt. The manual method does not allow for the processing of large volume of data on a regular and timely basis.

**Mechanical Method**

The mechanical method of data processing involves the use of machines such as typewriter, roneo machines, adding machines and the like. These machines facilitate human efforts in recording, classifying, manipulating, sorting and presenting data or information. The mechanical operations are basically routine in nature. There is virtually no creative thinking. The mechanical operations are noisy, hazardous, error prone and untidy. The mechanical method does not allow for the processing of large volume of data continuously and timely.

**Computer Method**

**The computer method of carrying out data processing has the following major features:**

a. Data can be steadily and continuously processed

b. The operations are practically not noisy

c. There is a store where data and instructions can be stored temporarily and permanently.

d. Errors can be easily and neatly corrected.

e. Output reports are usually very neat, decent and can be produced in various forms such as adding graphs, diagrams, pictures etc.

f. Accuracy and reliability are highly enhanced.

Below are further attributes of a computer which makes it to be an indispensable tool for human being:

**Characteristics of Computer**

**1. Speed:** The computer can manipulate large data at incredible speed and response time can be very fast.

**2. Accuracy:** Its accuracy is very high and its consistency can be relied upon. Errors committed in computing are mostly due to human rather than technological weakness. There are in-built error detecting schemes in the computer.

**3. Storage:** It has both internal and external storage facilities for holding data and instructions. This capacity varies from one machine to the other. Memories are built up in K(Kilo) modules where K = 1024 memory locations.

**4. Automatic:** Once a program is in the computer’s memory, it can run automatically each time it is opened. The individual has little or no instruction to give again.

**5. Reliability:** Being a machine, a computer does not suffer human traits of tiredness and lack of concentration. It will perform the last job with the same speed and accuracy as the first job every time even if ten million jobs are involved.

**6. Flexibility:** It can perform any type of task once it can be reduced to logical steps. Modern computers can be used to perform a variety of functions like on-line processing, multi-programming, real time processing etc.

**History of Computers**

The natural computing device man naturally has is the human brain. Man has an inbuilt capability to be able to carry out limited calculation, manipulation and express certain level of interpretations. The limitation of man in this capacity make him to think of devices that can be of assistance. Foremost of this device are Fingers and Toes, small stones and pegs.

The first early aids of counting were the fingers and toes. Man uses his fingers to help him count, it was a very simple device which help man to carry out simple additions. The fact that the total number that can be counted with fingers and toes is twenty make man to device another means like stones or small sticks

In the early days, when man wants to count say the number of animals he owned, he built a pile of stones with one stone for every animal. After he had finished building the piles he needed, he then needed to look at it and see how many there were on the piles he gathered. This method was used to count larger numbers. The stones could be used to count hundreds of numbers

A complete history of computing would include a multitude of diverse devices such as the ancient Chinese abacus, the Jacquard loom (1805) and Charles Babbage’s “analytical engine” (1834). It would also include discussion of mechanical, analog and digital computing architectures. As late as the 1960s, mechanical devices, such as the Marchant calculator, still found widespread application in science and engineering. During the early days of electronic computing devices, there was much discussion about the relative merits of analog vs. digital computers. In fact, as late as the 1960s, analog computers were routinely used to solve systems of finite difference equations arising in oil reservoir modeling. In the end, digital computing devices proved to have the power, economics and scalability necessary to deal with large scale computations. Digital computers now dominate the computing world in all areas ranging from the hand calculator to the supercomputer and are pervasive throughout society. Therefore, this brief sketch of the development of scientific computing is limited to the area of digital, electronic computers.

The evolution of digital computing is often divided into generations. Each generation is characterized by dramatic improvements over the previous generation in the technology used to build computers, the internal organization of computer systems, and programming languages. Although not usually associated with computer generations, there has been a steady improvement in algorithms, including algorithms used in computational science. The following history has been organized using these widely recognized generations as mileposts.

**First Generation Electronic Computers (1937 – 1953)**

Three machines have been promoted at various times as the first electronic computers. These machines used electronic switches, in form of vacuum tubes, instead of electromechanical relays. In principle the electronic switches were more reliable, since they would have no moving parts that would wear out, but technology was still new at that time and the tubes were comparable to relays in reliability. Electronic components had one major benefit, however: they could “open” and “close” about 1,000 times faster than mechanical switches.

The earliest attempt to build an electronic computer was by J. V. Atanasoff, a professor of physics and mathematics at Iowa State, in 1937. Atanasoff set out to build a machine that would help his graduate students solve systems of partial differential equations. By 1941, he and graduate student Clifford Berry had succeeded in building a machine that could solve 29 simultaneous equations with 29 unknowns. However, the machine was not programmable, and was more of an electronic calculator.

A second early electronic machine was Colossus, designed by Alan Turning for the British military in 1943. This machine played an important role in breaking codes used by the German army in World War II. Turning’s main contribution to the field of computer science was the idea of the Turing Machine, a mathematical formalism widely used in the study of computable functions. The existence of Colossus was kept secret until long after the war ended, and the credit due to Turning and his colleagues for designing one of the first working electronic computers was slow in coming.

The first general-purpose programmable electronic computer was the **Electronic Numerical Integrator and Computer** (ENIAC), built by J. Presper Eckert and John V. Mauchly at the University of Pennysylvania. Work began in 1943, funded by the Army Ordinance Department, which needed a way to compute ballistics during World War II. The machine wasn’t completed until 1945, but then it was used extensively for calculations during the design of the hydrogen bomb. By the time it was decommissioned in 1955 it had been used for research on the design of wind tunnels, random number generators, and weather prediction. Eckert, Mauchly, and John Von Neumann, a consultant to the ENIAC project, began work on a new machine before ENIAC was finished. The main contribution of EDVAC, their new project, was the notion of a stored program. There is some controversy over who deserves the credit for this idea, but no one knows how important the idea was to the future of general purpose computers. ENIAC was controlled by a set of external switches and dials; to change the program required physically altering the settings on these controls. These controls also limited the speed of the internal electronic operations. Through the use of a memory that was large enough to hold both instructions and data, and using the program stored in memory to control the order of arithmetic operations, EDVAC was able to run orders of magnitude faster than ENIAC. By storing instructions in the same medium as data, designers could concentrate on improving the internal structure of the machine without worrying about matching it to the speed of an external control.

Regardless of who deserves the credit for the stored program idea, the EDVAC project is significant as an example of the power of interdisciplinary projects that characterize modern computational science. By recognizing that functions, in the form of a sequence of instructions for a computer, can be encoded as numbers, the EDVAC group knew the instructions could be stored in the computer’s memory along with numerical data. The notion of using numbers to represent functions was a key step used by Goedel in his incompleteness theorem in 1937, work which Von Neumann, as a logician, was quite familiar with. Von Neumann’s background in logic, combined with Eckert and Mauchly’s electrical engineering skills, formed a very powerful interdisciplinary team. Software technology during this period was very primitive. The first programs were written out in machine code, i.e. programmers directly wrote down the numbers that corresponded to the instructions they wanted to store in memory. By the 1950s programmers were using a symbolic notation, known as assembly language, then hand translating the symbolic notation into machine code. Later programs known as assemblers performed the translation task.

As primitive as they were, these first electronic machines were quite useful in applied science and engineering. Atanasoff estimated that it would take eight hours to solve a set of equations with eight unknowns using a Marchant calculator, and 381 hours to solve 29 equations for 29 unknowns. The Atanasoff-Berry computer was able to complete the task in under an hour. The first problem run on the ENIAC, a numerical simulation used in the design of the hydrogen bomb, required 20 seconds, as opposed to forty hours using mechanical calculators. Eckert and Mauchly later developed what was arguably the first commercially successful computer, the UNIVAC; in 1952, 45 minutes after the polls closed and with 7% of the vote counted, UNIVAC predicted Eisenhower would defeat Stevenson with 438 electoral votes (he ended up with 442).

**Second Generation (1954 – 1962)**

The second generation saw several important developments at all levels of computer system design, from the technology used to build the basic circuits to the programming languages used to write scientific applications.

Electronic switches in this era were based on discrete diode and transistor technology with a switching time of approximately 0.3 microseconds. The first machines to be built with this technology include TRADIC at Bell Laboratories in 1954 and TX-0 at MIT’s Lincoln Laboratory. Memory technology was based on magnetic cores which could be accessed in random order, as opposed to mercury delay lines, in which data was stored as an acoustic wave that passed sequentially through the medium and could be accessed only when the data moved by the I/O interface.

Important innovations in computer architecture included index registers for controlling loops and floating point units for calculations based on real numbers. Prior to this accessing successive elements in an array was quite tedious and often involved writing self-modifying code (programs which modified themselves as they ran; at the time viewed as a powerful application of the principle that programs and data were fundamentally the same, this practice is now frowned upon as extremely hard to debug and is impossible in most high level languages). Floating point operations were performed by libraries of software routines in early computers, but were done in hardware in second generation machines.

During this second generation many high level programming languages were introduced, including FORTRAN (1956), ALGOL (1958), and COBOL (1959). Important commercial machines of this era include the IBM 704 and 7094. The latter introduced I/O processors for better throughput between I/O devices and main memory.

The second generation also saw the first two supercomputers designed specifically for numeric processing in scientific applications. The term “supercomputer” is generally reserved for a machine that is an order of magnitude more powerful than other machines of its era. Two machines of the 1950s deserve this title. The Livermore Atomic Research Computer (LARC) and the IBM 7030 (aka Stretch) were early examples of machines that overlapped memory operations with processor operations and had primitive forms of parallel processing.

**Third Generation (1963 – 1972)**

The third generation brought huge gains in computational power. Innovations in this era include the use of integrated circuits, or ICs (semiconductor devices with several transistors built into one physical component), semiconductor memories starting to be used instead of magnetic cores, microprogramming as a technique for efficiently designing complex processors, the coming of age of pipelining and other forms of parallel processing, and the introduction of operating systems and time-sharing.

The first ICs were based on small-scale integration (SSI) circuits, which had around 10 devices per circuit (or “chip”), and evolved to the use of medium-scale integrated (MSI) circuits, which had up to 100 devices per chip. Multilayered printed circuits were developed and core memory was replaced by faster, solid state memories. Computer designers began to take advantage of parallelism by using multiple functional units, overlapping CPU and I/O operations, and pipelining (internal parallelism) in both the instruction stream and the data stream. In 1964, Seymour Cray developed the CDC 6600, which was the first architecture to use functional parallelism. By using 10 separate functional units that could operate simultaneously and 32 independent memory banks, the CDC 6600 was able to attain a computation rate of 1 million floating point operations per second (1 Mflops). Five years later CDC released the 7600, also developed by Seymour Cray. The CDC 7600, with its pipelined functional units, is considered to be the first vector processor and was capable of executing at 10 Mflops. The IBM 360/91, released during the same period, was roughly twice as fast as the CDC 660. It employed instruction look ahead, separate floating point and integer functional units and pipelined instruction stream. The IBM 360-195 was comparable to the CDC 7600, deriving much of its performance from a very fast cache memory. The SOLOMON computer, developed by Westinghouse Corporation, and the ILLIAC IV, jointly developed by Burroughs, the Department of Defense and the University of Illinois, was representative of the first parallel computers. The Texas Instrument Advanced Scientific Computer (T I-ASC) and the STAR-100 of CDC were pipelined vector processors that demonstrated the viability of that design and set the standards for subsequent vector processors.

Early in this, third generation Cambridge and the University of London cooperated in the development of CPL (Combined Programming Language, 1963). CPL was, according to its authors, an attempt to capture only the important features of the complicated and sophisticated ALGOL. However, the ALGOL, CPL was large with many features that were hard to learn. In an attempt at further simplification, Martin Richards of Cambridge developed a subset of CPL called BCPL (Basic Computer Programming Language, 1967).

**Fourth Generation (1972 – 1984)**

The next generation of computer systems saw the use of large scale integration (LSI – 1000 devices per chip) and very large scale integration (VLSI – 100,000 devices per chip) in the construction of computing elements. At this scale entire processors will fit onto a single chip, and for simple systems the entire computer (processor, main memory, and I/O controllers) can fit on one chip. Gate delays dropped to about Ins per gate. Semiconductor memories replaced core memories as the main memory in most systems; until this time the use of semiconductor memory in most systems was limited to registers and cache. During this period, high speed vector processors, such as the CRAY 1, CRAY X-MP and CYBER 205 dominated the high performance computing scene.

Computers with large main memory, such as the CRAY 2, began to emerge. A variety of parallel architectures began to appear; however, during this period the parallel computing efforts were of a mostly experimental nature and most computational science was carried out on vector processors. Microcomputers and workstations were introduced and saw wide use as alternatives to time-shared mainframe computers.

Developments in software include very high level languages such as FP (functional programming) and Prolog (programming in logic). These languages tend to use a *declarative* programming style as opposed to the *imperative* style of Pascal, C. FORTRAN, et al. In a declarative style, a programmer gives a mathematical specification of what should be computed, leaving many details of how it should be computed to the compiler and/or runtime system. These languages are not yet in wide use, but are very promising as notations for programs that will run on massively parallel computers (systems with over 1,000 processors). Compilers for established languages started to use sophisticated optimization techniques to improve code, and compilers for vector processors were able to vectorize simple loops (turn loops into single instructions that would initiate an operation over an entire vector).

Two important events marked the early part of the third generation: the development of the C programming language and the UNIX operating system, both at Bell Labs. In 1972, Dennis Ritchie, seeking to meet the design goals of CPL and generalize Thompson’s B, developed the C language. Thompson and Ritchie then used C to write a version of UNIX for the DEC PDP-11. This C-based UNIX was soon ported to many different computers, relieving users from having to learn a new operating system each time they change computer hardware. UNIX or a derivative of UNIX is now a de facto standard on virtually every computer system.

An important event in the development of computational science was the publication of the Lax report. In 1982, the US Department of Defense (DOD) and National Science Foundation (NSF) sponsored a panel on Large Scale Computing in Science and Engineering, chaired by Peter D. Lax. The Lax Report stated that aggressive and focused foreign initiatives in high performance computing, especially in Japan, were in sharp contrast to the absence of coordinated national attention in the United States. The report noted that university researchers had inadequate access to high performance computers. One of the first and most visible of the responses to the Lax report was the establishment of the NSF supercomputing centers. Phase I on this NSF program was designed to encourage the use of high performance computing at American universities by making cycles and training on three (and later six) existing supercomputers immediately available. Following this Phase I stage, in 1984 – 1985 NSF provided funding for the establishment of five Phase II supercomputing centers. The Phase II centers, located in San Diego (San Diego supercomputing Centre); Illinois (National Center for Supercomputing Applications); Pittsburgh (Pittsburgh Supercomputing Center); Cornell (Cornell Theory Center); and Princeton (John Von

Neumann Center), have been extremely successful at providing computing time on supercomputers to the academic community. In addition they have provided many valuable training programs and have developed several software packages that are available free of charge. These Phase II centers continue to augment the substantial high performance computing efforts at the National Laboratories, especially the Department of Energy (DOE) and NASA sites.

**Fifth Generation (1984 – 1990)**

The development of the next generation of computer systems is characterized mainly by the acceptance of parallel processing. Until this time, parallelism was limited to pipelining and vector processing, or at most to a few processors sharing jobs. The fifth generation saw the introduction of machines with hundreds of processors that could all be working on different parts of a single program. The scale of integration in semiconductors continued at an incredible pace, by 1990 it was possible to build chips with a million components – and semiconductor memories became standard on all computers.

Other new developments were the widespread use of computer networks and the increasing use of single-user workstations. Prior to 1985, large scale parallel processing was viewed as a research goal, but two systems introduced around this time are typical of the first commercial products to be based on parallel processing. The Sequent Balance 8000 connected up to 20 processors to a single shared memory module (but each processor had its own local cache). The machine was designed to compete with the

DEC VAX-780 as a general purpose Unix system, with each processor working on a different user’s job. However, Sequent provided a library of subroutines that would allow programmers to write programs that would use more than one processor, and the machine was widely used to explore parallel algorithms and programming techniques. The Intel iPSC -1, nicknamed “the hypercube”, took a different approach. Instead of using one memory module, Intel connected each processor to its own memory and used a network interface to connect processors. This distributed memory architecture meant memory was no longer a bottleneck and large systems (using more processors) could be built. The largest iPSC-1 had 128 processors. Toward the end of this period, a third type of parallel processor was introduced to the market. In this style of machine, known as a *data-parallel* or SIMD, there are several thousand very simple processors. All processors work under the direction of a single control unit; i.e. if the control unit says “add a to b” then all processors find their local copy of a and add it to their local copy of b. Machines in this class include the Connection Machine from Thinking Machines, Inc., and the MP-1 from MasPar, Inc.

Scientific computing in this period was still dominated by vector processing. Most manufacturers of vector processors introduced parallel models, but there were very few (two to eight) processors in these parallel machines. In the area of computer networking, both wide area network (WAN) and local area network (LAN) technology developed at a rapid pace, stimulating a transition from the traditional mainframe computing environment towards a distributed computing environment in which each user has their own workstation for relatively simple tasks (editing and compiling programs, reading mail) but sharing large, expensive resources such as file servers and supercomputers. RISC technology (a style of internal organization of the CPU) and plummeting costs for

RAM brought tremendous gains in computational power of relatively low cost workstations and servers. This period also saw a marked increase in both the quality and quantity of scientific visualization.

**Sixth Generation (1990 to date )**

Transitions between generations in computer technology are hard to define, especially as they are taking place. Some changes, such as the switch from vacuum tubes to transistors, are immediately apparent as fundamental changes, but others are clear only in retrospect. Many of the developments in computer systems since 1990 reflect gradual improvements over established systems, and thus it is hard to claim they represent a transition to a new “generation”, but other developments will prove to be significant changes.

In this section, we offer some assessments about recent developments and current trends that we think will have a significant impact on computational science. This generation is beginning with many gains in parallel computing, both in the hardware area and in improved understanding of how to develop algorithms to exploit diverse, massively parallel architectures. Parallel systems now compete with vector processors in terms of total computing power and most especially parallel systems to dominate the future.

Combinations of parallel/vector architectures are well established, and one corporation (Fujitsu) has announced plans to build a system with over 200 of its high and vector processors. Manufacturers have set themselves the goal of achieving teraflops (1012 arithmetic operations per second) performance by the middle of the decade, and it is clear this will be obtained only by a system with a thousand processors or more. Workstation technology has continued to improve, with processor designs now using a combination of RISC, pipelining, and parallel processing. As a result it is now possible to procure a desktop workstation that has the same overall computing power (100 megaflops) as fourth generation supercomputers. This development has sparked an interest in heterogeneous computing: a program started on one workstation can find idle workstations elsewhere in the local network to run parallel subtasks.

One of the most dramatic changes in the sixth generation is the explosive growth of wide area networking. Network bandwidth has expanded tremendously in the last few years and will continue to improve for the next several years. T1 transmission rates are now standard for regional networks, and the national “backbone” that interconnects regional networks uses T3. networking technology is becoming more widespread than its original strong base in universities and government laboratories as it is rapidly finding application in K-12 education, community networks and private industry. A little over a decade after the warning voiced in the Lax report, the future of a strong computational science infrastructure is bright.

**Characteristics of computer**

The characteristics of computers that have made them so powerful and universally useful are speed, accuracy, diligence, versatility and storage capacity.

Speed: Computers work at an incredible speed. A powerful computer is capable of performing about 3-4 million simple instructions per second.

Accuracy: In addition to being fast, computers are also accurate. Errors that may occur can almost always be attributed to human error (inaccurate data, poorly designed system or faulty instructions/programs written by the programmer)

Diligence: Unlike human beings, computers are highly consistent. They do not suffer from human traits of boredom and tiredness resulting in lack of concentration. Computers, therefore, are better than human beings in performing voluminous and repetitive jobs.

Versatility: Computers are versatile machines and are capable of performing any task as long as it can be broken down into a series of logical steps. The presence of computers can be seen in almost every sphere – Railway/Air reservation, Banks, Hotels, Weather forecasting and many more.

Storage Capacity: Today’s computers can store large volumes of data. A piece of information once recorded (or stored) in the computer, can be retrieved almost instantaneously.

**The Computing System**

The computing system is made up of the computer system, the user and the environment in which the computer is operated.

**Computer Users**

Computer users are the different categories of personnel that operates the computer. We have expert users and casual users. The expert users could be further categorized into computer engineers, computer programmers and computer operators.

**The Computing Environment**

The computing environment ranges from the building housing the other elements of the computing system namely the computer and the users, the furniture, auxiliary devices such as the voltage stabilizer, the Uninterruptible Power Supply System (UPS), the fans, the air conditioners etc.

**The Computer System**

The computer system is made up of the hardware and the software.

**Hardware/Functional Components**

A computer system consists of mainly four basic units (See the fig. below); namely input unit, storage unit, central processing unit and output unit. Central Processing unit further includes Arithmetic & logic unit and control unit.

Central Processing unit

Control unit

Input Unit

Outputunit

ALU

Auxiliary storage

Memory

A computer performs five major operations or functions irrespective of its size and make. These are:

• it accepts data or instructions as input

• it stores data and instruction

• it processes data as per the instructions

• it controls all operations inside a computer

• it gives results in the form of output.

When a program is executing, the computer proceeds through a series of steps, the *fetch-execute cycle*:

1. The control unit retrieves (*fetches*) the next coded instruction from memory.

2. The instruction is translated into control signals.

3. The control signals tell the appropriate unit (arithmetic/logic unit, memory, I/O device) to perform (*execute*) the instruction. 4. The sequence repeats from Step 1.

a). Processing Unit: The task of performing operations like arithmetic and logical operations is called processing. The Central Processing Unit (CPU) takes data and instructions from the storage unit and makes all sorts of calculations based on the instructions given and the type of data provided. It is then sent back to the storage unit. CPU includes Arithmetic logic unit (ALU) and control unit (CU)

• Arithmetic Logic Unit: All calculations and comparisons, based on the instructions provided, are carried out within the ALU. It performs arithmetic functions like addition, subtraction, multiplication, division and also logical operations like greater than, less than and equal to etc.

• Control Unit: Controlling of all operations like input, processing and output are performed by control unit. It orders the step by step processing of all operations inside the computer.

b) Memory Unit: this is the medium of storage of a computer system. Computer’s memory can be classified into two types; primary memory and secondary memory

i). Primary Memory can be further classified as RAM and ROM.

• RAM or Random Access Memory is a unit of memory in a computer system. It is the place in a computer where the operating system, application programs and the data in current use are kept temporarily so that they can be accessed by the computer’s processor. It is said to be ‘volatile’ since its contents are accessible only as long as the computer is on. The contents of RAM are no longer available once the computer is turned off.

ROM or Read Only Memory is a special type of memory which can only be read and contents of which are not lost even when the computer is switched off. It typically contains manufacturer’s instructions. Among other things, ROM also stores an initial program called the ‘bootstrap loader’ whose function is to start the operation of computer system once the power is turned on.

ii. Secondary Memory

RAM is volatile memory having a limited storage capacity. Secondary/auxiliary memory is storage other than the RAM. These include devices that are peripheral and are connected and controlled by the computer to enable permanent storage of programs and data.

Secondary storage devices are of two types; magnetic and optical. Magnetic devices include hard disks and optical storage devices are CDs, DVDs, Pen drive, Zip drive etc.

• Hard Disk

Hard disks are made up of rigid material and are usually a stack of metal disks sealed in a box. The hard disk and the hard disk drive exist together as a unit and is a permanent part of the computer where data and programs are saved. These disks have storage capacities ranging from 1GB to 80 GB and more. Hard disks are rewritable.

• Compact Disk

Compact Disk (CD) is portable disk having data storage capacity between 650-700 MB. It can hold large amount of information such as music, full-motion videos, and text etc. CDs can be either read only or read/write type.

• Digital Video Disk

Digital Video Disk (DVD) is similar to a CD but has larger storage capacity and enormous clarity. Depending upon the disk type it can store several Gigabytes of data. DVDs are primarily used to store music or movies and can be played back on your television or the computer too.

c) Input/output Devices: These devices are used to enter information and instructions into a computer for storage or processing and to deliver the processed data to a user. Input/output devices are required for users to communicate with the computer. In simple terms, input devices bring information INTO the computer and output devices bring information OUT of a computer system. These input/output devices are also known as peripherals since they surround the CPU and memory of a computer system.

Input Devices an input device is any device that provides input to a computer. There are many input devices, but the two most common ones are a keyboard and mouse. Every key you press on the keyboard and every movement or click you make with the mouse sends a specific input signal to the computer.

• Keyboard: The keyboard is very much like a standard typewriter keyboard with a few additional keys. The basic layout of characters is maintained to make it easy to use the system. The additional keys are included to perform certain special functions. These are known as function keys that vary in number from keyboard to keyboard.

• Mouse: A device that controls the movement of the cursor or pointer on a display screen. A mouse is a small object you can roll along a hard and flat surface. Its name is derived from its shape, which looks a bit like a mouse. As you move the mouse, the pointer on the display screen moves in the same direction.

• Trackball: A trackball is an input device used to enter motion data into computers or other electronic devices. It serves the same purpose as a mouse, but is designed with a moveable ball on the top, which can be rolled in any direction.

• Touchpad: A touch pad is a device for pointing (controlling input positioning) on a computer display screen. It is an alternative to the mouse. Originally incorporated in laptop computers, touch pads are also being made for use with desktop computers. A touch pad works by sensing the user’s finger movement and downward pressure.

• Touch Screen: It allows the user to operate/make selections by simply touching the display screen. A display screen that is sensitive to the touch of a finger or stylus, monitors and industrial control panels.

• Light Pen: Light pen is an input device that utilizes a light-sensitive detector to select objects on a display screen.

• Magnetic ink character recognition (MICR): MICR can identify character printed with a special ink that contains particles of magnetic material. This device particularly finds applications in banking industry.

• Optical mark recognition (OMR): Optical mark recognition, also called mark sense reader is a technology where an OMR device senses the presence or absence of a mark, such as pencil mark. OMR is widely used in marking tests such as aptitude test.

• Scanner: Scanner is an input device that can read text or illustration printed on paper and translates the information into a form that the computer can use. A scanner works by digitizing an image.

Output Devices:

Output device receives information from the CPU and presents it to the user in the desired form. The processed data, stored in the memory of the computer is sent to the output unit, which then converts it into a form that can be understood by the user. The output is usually produced in one of the two ways – on the display device, or on paper (hard copy).

• Monitor: is often used synonymously with “computer screen” or “display.” Monitor is an output device that resembles the television screen. The monitor is associated with a keyboard for manual input of characters and displays the information as it is keyed in. It also displays the program or application output. Like the television, monitors are also available in different sizes.

• Printer: Printers are used to produce paper (commonly known as hardcopy) output. Based on the technology used, they can be classified as Impact or Non-impact printers.

Impact printers use the typewriting printing mechanism wherein a hammer strikes the paper through a ribbon in order to produce output. Dot-matrix and Character printers fall under this category.

Non-impact printers do not touch the paper while printing. They use chemical, heat or electrical signals to etch the symbols on paper. Inkjet, Deskjet, Laser, Thermal printers fall under this category of printers.

• Plotter: Plotters are used to print graphical output on paper. It interprets computer commands and makes line drawings on paper using multicoloured automated pens. It is capable of producing graphs, drawings, charts, maps etc.

• Facsimile (FAX): Facsimile machine, a device that can send or receive pictures and text over a telephone line. Fax machines work by digitizing an image.

• Sound cards and Speaker(s): An expansion board that enables a computer to manipulate and output sounds. Sound cards are necessary for nearly all CD-ROMs and have become commonplace on modern personal computers. Sound cards enable the computer to output sound through speakers connected to the board, to record sound input from a microphone connected to the computer, and manipulate sound stored on a disk.

**SOFTWARE**

Computer software is the set of programs that makes the hardware perform a set of tasks in particular order. Hardware and software are complimentary to each other. Both have to work together to produce meaningful results. Computer software is classified into two broad categories; system software and application software

1. Applications

Software that can perform a specific task for the user, such as word processing, accounting, budgeting or payroll, fall under the category of application software. Word processors, spreadsheets, database management systems are all examples of general purpose application software.

Types of application software are:

• Word processing software: The main purpose of this software is to produce documents. MS-Word, Word Pad, Notepad and some other text editors are some of the examples of word processing software.

• Database software: Database is a collection of related data. The purpose of this software is to organize and manage data. The advantage of this software is that you can change way data is stored and displayed. MS access, dBase, FoxPro, Paradox, and Oracle are some of the examples of database software.

• Spread sheet software: The spread sheet software is used to maintain budget, financial statements, grade sheets, and sales records. The purpose of this software is organizing numbers. It also allows the users to perform simple or complex calculations on the numbers entered in rows and columns. MS-Excel is one of the examples of spreadsheet software.

• Presentation software: This software is used to display the information in the form of slide show. The three main functions of presentation software is editing that allows insertion and formatting of text, including graphics in the text and executing the slide shows. The best example for this type of application software is Microsoft PowerPoint.

• Multimedia software: Media players and real players are the examples of multimedia software. This software will allow the user to create audio and videos. The different forms of multimedia software are audio converters, players, burners, video encoders and decoders

1. System software

System software consists of a group of programs that control the operations of a computer equipment including functions like managing memory, managing peripherals, loading, storing, and is an interface between the application programs and the computer. MS DOS (Microsoft’s Disk Operating System), UNIX are examples of system software.

1. Programmable Software

Computer language or programming language is a coded syntax used by computer programmers to communicate with a computer. Computer language establishes a flow of communication between software programs. The language enables a computer user to dictate what commands the computer must perform to process data. These languages can be classified into following categories.

1. Machine language

2. Assembly language

3. High level language

High Level Languages are user-friendly languages which are similar to English with vocabulary of words and symbols. These are easier to learn and require less time to write. They are problem oriented rather than ‘machine’ based. A compiler is a special program that processes statements written in a particular programming language called source code and converts them into machine language or “machine code” that a computer’s processor uses. Compiler translates high level language programs directly into machine language program. This process is called compilation.

**CLASSIFICATION OF COMPUTERS**

**Introduction**

The computer has passed through many stages of evolution from the days of the mainframe computers to the era of microcomputers. Computers have been classified based on different criteria. In this unit, we shall classify computers based on three popular methods.

**Objectives**

The objectives of this unit are to:

i. Classify computers based on size, type of signal and purpose.

ii. Study the features that differentiate one class of the computer from the others.

**Categories of Computers**

Although there are no industry standards, computers are generally classified in the following ways:

**Classification Based On Signal Type**

There are basically three types of electronic computers. These are the Digital, Analog and Hybrid computers.

**Digital Computer**

Represent its variable in the form of digits. It counts the data it deals with, whether representing numbers, letters or other symbols, are converted into binary form on input to the computer. The data undergoes a processing after which the binary digits are converted back to alpha numeric form for output for human use. Because of the fact that business applications like inventory control, invoicing and payroll deal with discrete values (separate, disunited, discontinuous); they are beset processed with digital computers. As a result of this, digital computers are mostly used in commercial and business places today.

**Analog Computer**

It measures rather than counts. This type of computer sets up a model of a system. Common type represents it variables in terms of electrical voltage and sets up circuit analog to the equation connecting the variables. The answer can be either by using a voltmeter to read the value of the variable required, or by feeding the voltage into a plotting device. They hold data in the form of physical variables rather than numerical quantities. In theory, analog computers give an exact answer because the answer has not been approximated to the nearest digit. Whereas, when we try to obtain the answers using a digital voltmeter, we often find that the accuracy is less than that which could have been obtained from an analog computer.

It is almost never used in business systems. It is used by the scientist and engineer to solve systems of partial differential equations. It is also used in controlling and monitoring of systems in such areas as hydrodynamics and rocketry; in production.

There are two useful properties of this computer once it is programmed:

1. It is simple to change the value of a constant or coefficient and study the effect of such changes.

2. It is possible to link certain variables to a time pulse to study changes with time as a variable, and chart the result on an X-Y plotter.

**Hybrid Computer**

In some cases, the user may wish to obtain the output from an analog computer as processed by a digital computer or vice versa. To achieve this, he set up a hybrid machine where the two are connected and the analog computer may be regarded as a peripheral of the digital computer. In such a situation, a hybrid system attempts to gain the advantage of both the digital and the analog elements in the same machine. This kind of machine is usually a special-purpose device which is built for a specific task. It needs a conversion element which accepts analog inputs, and output digital value. Such converters are called digitizers. There is need for a converter from analog to digital also. It has the advantage of giving real-time response on a continuous basis. Complex calculations can be dealt with by the digital elements, thereby requiring a large memory, and giving accurate results after programming. They are mainly used in aerospace and process control applications.

**Classification By Purpose**

Depending on their flexibility in operation, computers are classified as either special purpose or general purpose.

**Special Purpose Computers**

A special purpose computer is one that is designed to solve a restricted class of problems. Such computers may even be designed and built to handle only one job. In such machines, the steps or operations that the computer follows may be built into the hardware. Most of the computers used for military purposes fall into this class. Other example of special purpose computers include:

* Computers designed specifically to solve navigational problems.
* Computers designed for tracking airplane or missiles.
* Computers used for process control applications in industries such as oil refinery, chemical manufacture, steel processing and power generation.
* Computers used as robots in factories like vehicles assembly plants and glass industries.

**General Attributes of Special Purpose Computers**

Special purpose computer are usually very efficient for the tasks for which they are specially designed.

They are very much less complex than the General-Purpose Computers. The simplicity of the circuiting stems from the fact that provision is made only for limited facilities. They are very much cheaper than the General-Purpose type since they involve less components and are less complex.

**General-Purpose Computers**

General-Purpose computers are computers designed to handle wide range of problems.

Theoretically, a general-purpose computer can be adequate by means of some easily alterable instructions to handle any problems that can be solved by computation. In practice however, there are limitations imposed by memory size, speed and the type of input/output devices. Examples of areas where the general purpose are employed include the following:

· Payroll

· Banking

· Billing

· Sales analysis

· Cost accounting

· Manufacturing scheduling

· Inventory control

General Attributes of General-Purpose Computers

* General-Purpose computers are more flexible than special purpose computers.
* They can handle a wide spectrum of problems.

They are less efficient than the special-purpose computers due to such problems as;

* Inadequate storage;
* Low operating speed;
* Coordination of the various tasks and subsection may take time.
* General Purpose Computers are more complex than the special purpose ones.

**Classification of Computers According to Capacity**

In the past, the capacity of computers was measured in terms of physical size. Today, however, the physical size is not a good measure of capacity because the modern technology has made it possible to achieve compactness. A better measure of capacity today is the volume of work that computer can handle. The volume of work that a given computer handles is closely tied to the cost and to the memory size of computer. Therefore, most authorities today accept the price of rental price as the standard for ranking computers. Here, both memory size and cost shall be used to rank (classify) computer into three main categories as follows:

(a)Microcomputers

(b)Medium/Mini/Small Computers

(c)Large Computer/Main Frames.

**Micro Computers**

Microcomputers, also known as single board computers, are the cheapest class of computers. In the microcomputer, we do not have a Central Processing Unit (CPU) as we have in the larger computers rather we have a microprocessor chip as the main data processing unit. They are the cheapest smallest and can operate under normal office condition. Examples are IBM, APPLE, COMPAQ, Hewlett Packard (HP), Dell Toshiba, e.t.c.

**Different Types of Personal Computers (Micro Computers)**

Normally, personal computers are placed on table desk hence they are referred to as desktop personal computers. Still other types are available under the categories of personal computers. They are:

**Laptop Computers** are small size types that are battery-operated. The screen is used to cover the system while the keyboard is installed flatly on the system unit. They could be carried about like a box when closed after operation and can be operated in vehicles while on a journey.

**Notebook Computer**

This is like laptop computers but smaller in size. Though small, it comprises all the components of a full system.

**Palmtop Computer**

Palmtop computer is far smaller in size. All the components are complete as any of the above but made smaller so that it can be held on the palm.

**Uses of Personal Computers**

Personal computers can perform the following functions:

* Can be used to produce documents like memos, reports, letters and briefs.
* Can be used to calculate budget and accounting tasks
* It can analyze numeric function
* It can create illustrations
* Can be used for electronic mails
* Can help in making schedule and plan projects.
* It can assist in schedules and plan projects.
* It can assist in searching for specific information from lists or from reports.

**Advantages of Personal Computers**

* Computer is versatile; it can be used in any establishment.
* Has faster speed for processing data.
* Can deal with several data at a time
* Can attend to several users at the same time, thereby able to process several jobs at a time.
* Capable of storing several data.
* Operating of Computer is less fatigue
* Network possible, that is linking of two or more computers together.

**Disadvantages of Personal Computers**

* Computer is costly to maintain.
* It is very fragile and complex to handle
* It requires special skill to operate
* With the invention and innovation everyday, computer suffers from being obsolete.
* It can lead to unemployment when used mostly in less Developed Countries.
* Some computers cannot function properly without the aid of cooling system e.g. air-condition or fan in some locations.

**Mini Computers**

The Mini Computers have memory capacity in the range 128K bytes to 256 Kbytes and are also not expensive but reliable and smaller in size compare to mainframe. It was first introduced in 1965; when DEC (Digital Equipment Corporation) built the PDP – 8.

Other Mini Computer includes WANG VS.

**Mainframe**

The Main Frame Computers often called number crunches have memory capacity of the order of 4 Kbytes and they are very expensive. They can execute up to 100MIPS (Meanwhile Instructions Per Second). They have large systems and are used by many people for a variety of purpose.

**COMPUTER HARDWARE**

In this module we shall discuss the following:

Hardware components – (the front, back and inside of the system unit)

The peripheral devices

The Auxiliary equipment

**Introduction**

Your Personal Computer (PC) is really a collection of separate items working together as a team-with you as the captain. Some of these components are essential; others simply make working more pleasant or efficient. Adding extra items expands the variety of tasks you can accomplish with your machine.

**The Objectives**

The objectives of this unit are to:

i. Familiarize the student with the components of the computer.

ii. Enable the student to appreciate the importance of each of the components to the overall smooth operations of the computer.

**The System Unit**

The system unit is the main unit of a PC. It is the Computer itself while other units attached to it are regarded as peripherals. It could be viewed as the master conductor orchestrating your PC’s operation. It is made up of several components like the Motherboard, Processor, Buses, memory, power supply unit, etc. This unit (system unit) has been confused over the years by novices as the CPU. This is not true. The CPU (Central Processing Unit) or simply processor is a component within the system unit and it is not the only thing that makes up the system unit. Hence, it will be wrong to equate the system unit with the CPU.

**Front of the System Unit**

**Lights**

Your unit may display a variety of colored light on the front panel, including power and turbo signals, and light to indicate if the Hard or Floppy disk are being read or written to.

**Key Lock**

You can stop intruders tampering with your PC by using the Lock on the front panel. Turning the key prevents the key board from working.

**Turbo Button**

Some PCs offer a choice of speeds at which they can run. A turbo switch is usually left so the computer runs at its fastest speed.

**Reset Button**

If your PC “freezes” and won’t respond to any command, try starting it up again using the reset button. Pressing the reset button loses all the work you have not saved in that session, so use it only as a last resort.

**Power On/Off**

All PCs have main power switch on the system unit. Sometimes this control is placed on the outside back panel.

**Floppy Disk Drives**

Either, or both, of two standard types of floppy disk drive may be found at the front of the system unit. Some systems also have internal CD-ROM or tape drives.



CD-ROM or DVD drive

**Back of the System Unit**

**Fan Housing**

The electronic components in your PC generate a lot of heat. To prevent overheating, a fan at the back of the unit removes hot air from the system.

**Power “in” and “out” Sockets**

Cables plugged into these sockets carry power from the electrical outlet to the system unit and from the system unit to the monitor.

**Joystick Port**

Using a joystick is often much better than pressing keys to control movements when playing a computer game.

**Serial Ports**

Serial Ports often connect the PC to modem or mouse. Most PCs are fitted with two

serial ports that may be labeled “S101” and “S102”, “Serial 1” and “Serial 2”, or “COM 1” and “COM 2”.

**Sound Jacks**

If you have a sound fitted inside your system unit, you will see a jack or jacks at the back. These can be used to connect your PC to speakers, a microphone, or an eternal sound source.

**Keyboard Port**

The cable from your keyboards ends with a round connector, which plugs into the keyboard port.

**Network Adapter**

If an expansion card is fitted to link your PC with other PCs in your office you will see a network connector at the back of the system unit.

**Monitor Port**

A cable from your monitor plugs into this port and carries display information to the monitor.

**Bays for Expansion Cards**

PCs are easily expanded-perhaps to provide a modem, sound or faster graphics. You can plug cards into expansion slots inside the PC. The end of an expansion card shows at the back of your machine, allowing you to connect items.



**Inside the system unit**

The brain behind everything that happens in your PC is contained within the system unit. Inside the unit are the impressive electronics that run programs, handle instructions, and determine the results. Most of the more important items are described below:

**Battery**

A small battery powers a clock to keep track of the time when the PC is turned off. It also maintains low electricity to certain RAM chips that record which components are installed.

**Disk Drive Controller Card**

This card controls the PC’s disk drive motors and transfers data. The serial and parallel ports at the back of the card link internal PC components with external devices such as mouse and printer.

**Display Adapter Card (Video Card)**

All the information your computer will display is stored in its memory. To be useful, you need to see the information. The display adapter card is the link between the PC’s memory and the monitor.

**Expansion Slots**

These long narrow connectors allow you to plug in expansion cards (also known as adapter cards), which offer extra options not available on a basic PC.

**ROM Chips**

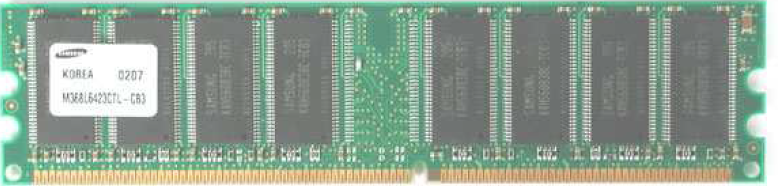
Read-only memory (ROM) chips have data written on them during manufacturing that tells the CPU what to do when the PC is switched on. The data is always there, even when you switch the PC off.

**RAM Chips**

When a computer is switched on and running a program, RAM (Random Access Memory) is used for purposes such as holding the program and its data. But when the PC is switched off, anything held in RAM is lost.

**Empty RAM Chip Slots**

These slots let you expand your computer’s memory by adding extra RAM chips or modules. Some PC’s work even faster because they come equipped with Cache Memory. Cache Memory consists of expensive and very fast memory chips that store the data or instructions that the CPU will look at next. Cache memory can speed up work on your computer enormously.



RAM chip

**Central Processing Unit (CPU)**

****

Intel Processor

The Microprocessor, or Central Processing Unit (CPU), is the computer’s most important single item. It does all the PC’s thinking and runs the programs (series of instructions) that you request.

**CPU Support Chips**

These chips help the CPU manage all the other parts of the computer.

**Math Coprocessor Slot**

A math coprocessor, present in some PCs, assists the CPU in its number-crunching activities (if programs have been designed to use it).



CPU fan

**Speaker**

The speaker emits the computer’s sound output.

**Power Supply Unit**

All the components in a PC need electrical supply. Most need a 5-volt supply although the floppy disk drive motors required 12 volts. If the components were connected to normal household current, they would blow up, so the power supply unit converts high voltage electrical current to a low voltage.

**Hard Disk Drive**

The hard disk is your computer’s main permanent storage unit, holding large amount of data and programs. Unlike data held in RAM, the information on the hard disk is not affected when the PC is turned off – it remains there unless you instruct the PC to overwrite it or the hard disk is damaged.



Hard drive (Hard disk)

**Motherboard**

All the electronic components in a PC are mounted on a piece of fiberglass called the motherboard. Fiberglass cannot conduct electricity, so each component is insulated from all the others. Thin lines of metal on the surface of the fiberglass connect pins from one component to another, forming the computer’s electrical circuits.

****

Components of a motherboard

**Intel CPUs**

The earliest PCs were equipped with a CPU from Intel Corporation called the 8088. The next generation of PCs used CPU known by the number “80286 and were called “PC/AT” computers. Subsequently, PCs have been supplied with more and more powerful CPUs – the 80386, the 80486, and the more recent and impressive of all, the Intel Pentium (I, II, III, IV& M).

All these PC processors belong to a family called 80 x 86. In general, you can run the same software on PCs containing different CPUs within this family. From the outside, the chips look different only in sizes and number of pin-put inside, an 80486 has over one million components to the 3,500 that were in the first 8088. Because of these differences, the latest Pentiums runs over ten times faster.

**What is CPU**

The CPU is certainly the most important PC component. CPU stands for *Central Processing Unit*. Let us briefly study that name:

* It is a processor, because it processes (moves and calculates) data.
* It is central, because it is the center of PC data processing.
* It is a unit, because it is a chip, which contains millions of transistors.

**CPU Speed**

The speed of a CPU is measured in megahertz (MHz). A computer has central clock that keeps all the components in time with each other; one hertz is similar to a clock tick and megahertz is equal to one million ticks per second. If your PC runs at 333 or 400MHz, the central clock ticks 333 or 400 million times every second. As you might imagine, the faster the clock ticks, the faster the computer runs. Without the CPU, there would be no PC. Like all other hardware components, the CPUs are continually undergoing further development. You can see the explosive technological development in data processing most clearly in the development of newer and faster CPUs. The CPUs have for years doubled their performance about every 18 months and there are no indications that this trend will stop. When we now look at all the CPUs from a broader perspective, we can see that:

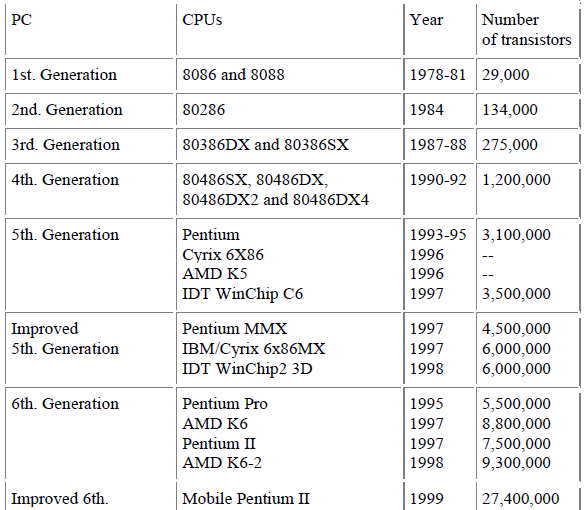
* The CPU history is closely tied to the companies IBM and especially *Intel*.
* The CPUs have their roots back to Intel's chip *4004* from 1971.
* The *compatibility* concept has been important throughout the development.

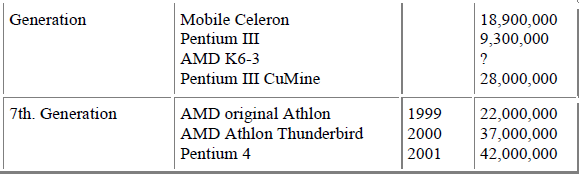
**Generations of CPUs**

There are CPUs of many brand names (IBM, Texas, Cyrix, AMD), and often they make models which overlap two generations. This can make it difficult to keep track of CPUs. Here is an attempt to identify the various CPUs according to generation:

**History of CPU**

The following table shows the different CPU *generations*.







Intel Processor

**DISKS**

**Floppy Disks**

Computers use disk to store information. Although there is a permanent hard disk that lives inside the system unit, you can use floppy disks to store and move data easily from one PC to another. Floppy disks come in two sizes, either 5¼ or 3½ inches in diameter. The smaller disks are able to store more data and are also less easily damaged, because of their thicker plastic cases. As both sizes can be either “high” or “low” capacity (or density), there are four main varieties of disks available. High-capacity disks are more expensive, but they can store much more information. Low-capacity disks are generally labeled DS/DD, which stands for “double sided/double density”, while the high-capacity floppy disks are labeled DS/HD (double sided/high-density”).

**Caring for Disks**

Treat floppy disks carefully, and you can take them almost anywhere safely. Don’t leave the disks in your PC when you finish a session. Also avoid putting anything heavy on top of your disks or leaving them in extremes of hot or cold temperature. Try not to carry disks loose in pockets or handbags where dust and dirt may get inside the containers. Take care to store them vertically, preferably in a special storage box.

Remember too that you should keep floppy disks away from magnetic fields, including hidden magnets such as those in telephone, radio and television speakers, amplifiers, desk fans, and photocopiers. If you do leave floppy disks near magnetic field, your data may become corrupted and will no longer be usable.

**Write Protecting Disks**

Write – protecting a disk means that you prevent the computer from erasing or writing over important data or programs that are already there. However, the PC can still read a write-protected disk.

**Computer Programming**

Computer programming (often shortened to programming, scripting, or coding) is the process of designing, writing, testing, debugging, and maintaining the source code of computer programs. This source code is written in one or more programming languages (such as C++, C#, Java etc.). The purpose of programming is to create a set of instructions that computers use to perform specific operations or to exhibit desired behaviors. The process of writing source code often requires expertise in many different subjects, including knowledge of the application domain, specialized algorithms and formal logic.

How Do We Write a Program? (Programming process)

A computer is not intelligent. It cannot analyze a problem and come up with a solution. A human (the *programmer*) must analyze the problem, develop the instructions for solving the problem, and then have the computer carry out the instructions. What’s the advantage of using a computer if it cannot solve problems? Once we have written a solution for the computer, the computer can repeat the solution very quickly and consistently, again and again. The computer frees people from repetitive and boring tasks.

To write a program for a computer to follow, we must go through a three-phase process: *problem solving* and *implementation*  and  *maintenance* see the Fig. below

Problem solving phase Implementation Phase

Problem analysis and

Specification

General solution

(Algorithm and

flowchart)



Verification

Concrete Solution

(Program)

Testing



Maintenance Phase

Problem-Solving Phase

1. Analysis and Specification. Understand (define) the problem and what the solution must do.

2. General Solution (Algorithm). Specify the required data types and the logical sequences of steps that solve the problem.

3. Verification. Follow the steps exactly to see if the solution can really solve the problem.

Implementation Phase

1*. Concrete Solution (Program).* Translate the algorithm (the general solution) into a programming language. Choice of appropriate programming language is a crucial issue in programming. This is because the required in-built function that can make the programming easier might be missing if the language of choice is not appropriate.

2*.* Testing. Run the computer programs, and then manually check the results. If you find errors, analyze the program and the algorithm to determine the source of the errors, and then make corrections. Once a program has been written correctly, it enters a third phase: *maintenance*.

Maintenance Phase

1. This is the actual phase where the program develop will be used to solve the problem for which it was design.

2. Maintain/Modify the program to meet changing requirements or to correct any errors that show up while using it.

**Programming Language:** A programming language is a simplified form of English (with math symbols) that adheres to a strict set of grammatical rules. English is far too complicated and ambiguous for today’s computers to follow. Programming languages, because they limit vocabulary and grammar, are much simpler. Although a programming language is simple in form, it is not always easy to use. Try giving someone directions to the nearest airport using a limited vocabulary of no more than 25 words, and you begin to see the problem. Programming forces you to write very simple and exact instructions.

Translating an algorithm into a programming language is called *coding* the algorithm. The products of the translation—the code for all the algorithms in the problem—are tested by collecting them into a program and running (*executing*) the program on the computer. If the program fails to produce the desired results, the programmer must debug it—that is, determine what is wrong and then modify the program, or even one or more of the algorithms, to fix it. The combination of coding and testing the algorithms is called *implementation*.

Code is the product of translating an algorithm into a programming language. The term *code* can refer to a complete program or to any portion of a program. There is no single way to implement an algorithm. For example, an algorithm can be translated into more than one programming language. Each translation produces a different implementation. Even when two people translate an algorithm into the same programming language, they are likely to come up with different implementations. Why? Because every programming language allows the programmer some flexibility in how an algorithm is translated. Given this flexibility, people adopt their own *styles* in writing programs, just as they do in writing short stories or essays.

Some people try to speed up the programming process by going directly from the problem definition to coding the program. Developing a general solution before you write a program helps you manage the problem, keep your thoughts straight, and avoid mistakes. If you don’t take the time at the beginning to think out and polish your algorithm, you spend a lot of extra time debugging and revising your program. So *think first, plan your solution and then coding*

Documentation: this is the written text and comments that make a program easier for others to understand, use, and Modify. In addition to solving the problem, implementing the algorithm, and maintaining the program, writing documentation is an important part of the programming process. Documentation includes written explanations of the problem being solved and the organization of the solution, comments embedded within the program itself, and user manuals that describe how to use the program. Many different people are likely to work on a program over a long period of time. Each of those people must be able to read and understand the code.

ALGORITHM

An algorithm is a step-by-step procedure for calculations. More precisely, an algorithm is an effective method expressed as a finite listof well-defined instructions for solving a problem.Starting from an initial state and initial input, the instructions describe actions that, when executed, will proceed through a finite number of well-defined successive states, eventually producing "output" and terminating at a final ending state.

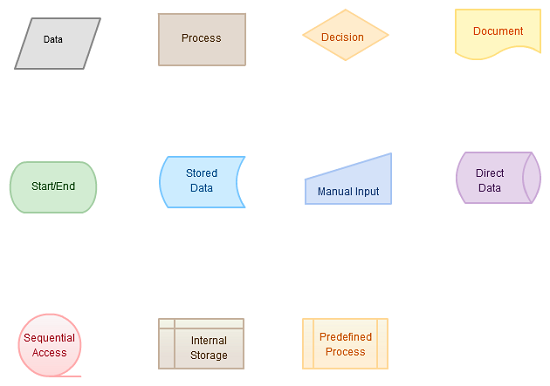
A flowchart is a type of diagram that represents an algorithm or process, showing the steps as boxes of various kinds, and their order by connecting them with arrows. This diagrammatic representation can give a step-by-step solution to a given problem. Process operations are represented in these boxes, and arrows connecting them represent flow of control. Arrows implied sequence of operations. Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields.

Flowchart Symbols

In general, a flowchart is a visual representation, which shows you a sequence of operations that are to be performed in order to get the solution to a problem.

Described below are standard symbols along with a visual representation right below:

1. Data object - The Data object, often referred to as the I/O Shape shows the Inputs to and Outputs from a process.
2. Rectangle - This is used to represent an event which is controlled within the process. Typically this will be a step or action which is taken.
3. Diamond - Used to represent a decision point in the process. Typically, the statement in the symbol will require a `yes’ or `no’ response and branch to different parts of the flowchart accordingly.
4. Document - The Document object is a rectangle with a wave-like base. This shape is used to represent a Document or Report in a process flow.
5. Rounded box – This is used to represent an event which occurs automatically. Such an event will trigger a subsequent action, for example `receive telephone call, or describe a new state of affairs.
6. Stored data - This is a general data storage object used in the process flow as opposed to data which could be also stored on a hard drive, magnetic tape, memory card, of any other storage device.
7. Manual input - This object is represented by rectangle with the top sloping up from left to right. The Manual Input object signifies an action where the user is prompted for information that must be manually input into a system.
8. Direct data – Direct data object in a process flow represents information stored which can be accessed directly. This object represents a computer’s hard drive.
9. Circle - Used to represent a point at which the flowchart connects with another process. The name or reference for the other process should appear within the symbol.
10. Internal storage – This is an object which is commonly found in programming flowcharts to illustrate the information stored in memory, as opposed to on a file.
11. Predefined process – This allows you to write one subroutine and call it as often as you like from anywhere in the code.



It must also be stated that at this point in time that while learning the various symbols that are associated with flowcharts are rather important, you need to also remember that there are certain guidelines in flowcharting that deserves some respect as well. The following are some guidelines in flowcharting:

1. Proper Form is Essential: In drawing a proper flowchart, all necessary requirements should be listed out in logical order.
2. Clarity is Paramount: The flowchart should be clear, neat and easy to follow. There should not be any room for ambiguity in understanding the flowchart.
3. Stick to the Right Direction: The usual direction of the flow of a procedure or system is from left to right or top to bottom.
4. Standard for Flow Lines: Ideally just one flow line should come out from a process symbol.  While only one flow line should enter a decision symbol, around three flow lines (depending on the answer) should leave the decision symbol. Additionally, only one flow line is utilized together with a terminal symbol.
5. Logic precedes everything: If you are dealing with a complex flowchart then use connector symbols to minimize the number of flow lines. Ditch the intersection of flow lines to ensure effectiveness and better communication. It is imperative that your flowchart has a logical start and finish.

Sample Problems: Suppose a programmer needs an algorithm to determine an employee’s weekly wages. The programmer begins the programming process by analyzing the problem, breaking it into manageable pieces, and developing a general solution for each piece called an algorithm The algorithm reflects what would be done by hand:

1. Look up the employee’s pay rate.

2. Determine the hours worked during the week.

3. If the number of hours worked is less than or equal to 40, multiply the hours by the pay rate to calculate regular wages.

4. If the number of hours worked is greater than 40, multiply 40 by the pay rate to calculate regular wages, and then multiply the difference between the hours worked and 40, by the overtime pay rate to calculate overtime wages.

5. Add the regular wages to the overtime wages (if any) to determine total wages for the week.

**STATEMENTS**

In computer programming a statement is the smallest standalone element of an imperative programming language. A program written in such a language is formed by a sequence of one or more statements. A statement will have internal components (e.g., expressions).

### Simple statements

* Assignment statement: e.g., A:= A + 5
* Call statement: e.g. CLEARSCREEN()
* Return statement : return 5;
* goto: goto 5

Compound statement: contain more than one statements

If-statement: if A > 3 then WRITELN(A) else WRITELN("NOT YET"); end

|  |
| --- |
|  |

CONTROL STRUCTURES

Programming languages require that we use certain *control structures* to express algorithms as source code. There are four basic ways of structuring statements (instructions) in most programming languages:

* sequence, is a series of statements that are executed one after another
* selection, the conditional control structure, executes different statements depending on certain conditions
* loop, The repetitive control structure, , repeats statements while certain conditions are met and
* Subprograms,allows breaking of code into smaller units.

Each of these ways of structuring statements controls the order in which the computer executes the statements, which is why they are called control structures. Assume you’re driving a car. Going down a straight stretch of road is like following *sequence* of instructions. When you come to a T junction, on the road, you must decide which way to go and then take one or the other branch of the junction. This is what the computer does when it encounters a *selection control structure* (sometimes called a *branch* or *decision*) in a program. Sometimes you have to go around the block several times to find a place to park. The computer does the same sort of thing when it encounters a *loop* in a program. However the default setting is sequential.

Sequence selection loop

statement

statement

statement

statement

statement

statement

statement

statement

statement

decision

statement

statement

statement

statement

decision

statement

decision

statement

statement

decision

statement

statement

statement

statement

decision

statement

statement

statement

Arrays

Most mathematical and computer languages have some notation for repeating or order related values. These repeated structures are often called a matrix, a dimension, or a table.

When to use arrays

It makes sense to use arrays when there are repetitive values that are related and the programmer needs to iterate through most of them. The combination of arrays and do loops in the data step lend incredible power to programming. The fact that the variables in the array do not need to be related or even contiguous makes them even more convenient to use.

Subscripts

In programming, subscript is a symbol or number used to identify an element in an array. Usually, the subscript is placed in brackets following the array name. For example, AR[5] or AR(5) identifies element number 5 in an array called AR.

If the array is multidimensional, you must specify a subscript for each dimension. For example, MD[5][3][9] or MD(5,3,9) identifies an element in a three-dimensional array called MD.

Different programming languages have different rules for specifying subscripts. For example, the BASIC language uses parentheses in place of brackets.

In word processing, a character that appears slightly below the line, as in this example: H2. A superscript is a character that appears slightly above the line.

An array is a collection of like variables that share a single name. The individual elements of an array are referenced by appending a *subscript*, in square brackets, behind the name. The subscript itself can be any legitimate programming expression that yields an integer value, even a general expression. Therefore, arrays in C programming language for instance may be regarded as collections of like variables. Although arrays represent one of the simplest data structures, it has wide-spread usage in embedded systems. In general we allow random access to individual array elements. On

*Array Subscripts*

When an array element is referenced, the subscript expression designates the desired element by its position in the data. The first element occupies position zero, the second position one, and so on. It follows that the last element is subscripted by [N-1] where N is the number of elements in the array. The statement

data[9] = 0;

for instance, sets the tenth element of data to zero.

If the array has two dimensions, then two subscripts are specified when referencing. As programmers we may assign logical meaning to the first and second subscripts. For example we could consider the first subscript as the row and the second as the column. Then, the statement

y = AR[3][5];

Copies the information from the 4th row 6th column into the variable y. If the array has three dimensions, then three subscripts are specified when referencing. Then, the statement

humidity[2][3][4]=100;

**Introduction To BASIC programming Language**

BASIC is a family of general-purpose, high-level programming languages whose design philosophy emphasizes ease of use; the name is an acronym from Beginner's All-purpose Symbolic Instruction Code.

The original Dartmouth BASIC was designed in 1964 by John George Kemeny and Thomas Eugene Kurtz at Dartmouth College in New Hampshire, USA to provide computer access to non-science students. At the time, nearly all use of computers required writing custom software, which was something only scientists and mathematicians tended to do. The language and its variants became widespread on microcomputers in the late 1970s and 1980s, when it was typically a standard feature, and often part of the firmware of the machine. The presence of an easy-to-learn language such as BASIC on these early personal computers allowed small business owners to develop their own custom application software, leading to widespread use of these computers in businesses that previously did not have access to computing technology.

BASIC remains popular in numerous dialects and new languages influenced by BASIC such as Microsoft Visual Basic. In 2006, 59% of developers for the .NET Framework used Visual Basic .NET as their only programming language.

|  |
| --- |
|  |

### Typical BASIC keywords

Data manipulation

* LET: assigns a value (which may be the result of an [expression](http://en.wikipedia.org/wiki/Expression_%28programming%29)) to a variable.
* DATA: holds a list of values which are assigned sequentially using the READ command.

Program flow control

* IF ... THEN ... ELSE: used to perform comparisons or make decisions.
* FOR ... TO ... {STEP} ... NEXT: repeat a section of code a given number of times. A variable that acts as a counter is available within the loop.
* DO ... LOOP {WHILE} or {UNTIL}: repeat a section of code While/Until the specified condition is true. The condition may be evaluated before each iteration of the loop, or after.
* GOTO: jumps to a numbered or labeled line in the program.
* GOSUB: temporarily jumps to a numbered or labeled line, returning to the following line after encountering the RETURN Command. This is used to implement subroutines.
* ON ... GOTO/GOSUB: chooses where to jump based on the specified conditions.

Input and output

* PRINT: displays a message on the screen or other output device.
* INPUT: asks the user to enter the value of a variable. The statement may include a prompt message.

Miscellaneous

* REM: holds a programmer's comment or REMark; often used to give a title to the program and to help identify the purpose of a given section of code.

### Data types and variables

Minimal versions of BASIC had only integer variables and one-letter variable names. More powerful versions had floating-point arithmetic, and variables could be labeled with names six or more characters long.

String variables are usually distinguished in many microcomputer dialects by having $ suffixed to their name, and string values are typically enclosed in quotation marks.

Arrays in BASIC could contain integers, floating point or string variables.

Some dialects of BASIC supported matrices and matrix operations, useful for the solution of sets of simultaneous linear algebraic equations. These dialects would support matrix operations such as assignment, addition, multiplication (of compatible matrix types), and evaluation of a determinant. Microcomputer dialects often lacked this data type and required a programmer to provide subroutines to carry out equivalent operations.

PROBLEM-SOLVING TECHNIQUES

You solve problems every day, often unaware of the process you are going through. In a learning environment, you usually are given most of the information you need: a clear statement of the problem, the necessary input, and the required output. In real life, the process is not always so simple. You often have to define the problem yourself and then decide what information you have to work with and what the results Should be.

After you understand and analyze a problem, you must come up with a solution—an algorithm. Earlier we defined an algorithm as a step-by-step procedure for solving a problem in a finite amount of time with a finite amount of data. Although you work with algorithms all the time, most of your experience with them is in the context of *following* them. You follow a recipe, play a game, assemble a toy, or take medicine. In the problem-solving phase of computer programming, you will be *designing* algorithms, not following them. This means you must be conscious of the strategies you use to solve problems in order to apply them to programming problems.

1)Ask Questions

If you are given a task orally, you ask questions When? Why? Where? until you understand exactly what you have to do. If your instructions are written, you might put question marks in the margin, underline a word or a sentence, or in some other way indicate that the task is not clear. Your questions may be answered by a later paragraph, or you might have to discuss them with the person who gave you the task.

These are some of the questions you might ask in the context of programming:

• What do I have to work with—that is, what are my data?

• What do the data items look like?

• What are the operations to be performed on the data?

• How much data is there?

• How will I know when I have processed all the data?

• What should my output look like?

• What special error conditions might come up?

Look for Things That Are Familiar Never reinvent the wheel. If a solution exists, use it. If you’ve solved the same or a similar problem before, just repeat your solution. People are good at recognizing similar situations. We don’t have to learn how to go to the store to buy milk, then to buy eggs, and then to buy candy. We know that going to the store is always the same; only what we buy is different.

In programming, certain problems occur again and again in different guises. A good programmer immediately recognizes a subtask he or she has solved before and plugs in the solution. For example, finding the daily high and low temperatures is really the same problem as finding the highest and lowest grades on a test. You want the largest and smallest values in a set of numbers

2. Solve by Analogy

Often a problem reminds you of one you have seen before. You may find solving the problem at hand easier if you remember how you solved the other problem. In other words, draw an analogy between the two problems. For example, a solution to a perspective projection problem from an art class might help you figure out how to compute the distance to a landmark when you are on a cross-country hike. As you work your way through the new problem, you come across things that are different than they were in the old problem, but usually these are just details that you can deal with one at a time.

Analogy is really just a broader application of the strategy of looking for things that are familiar. When you are trying to find an algorithm for solving a problem, don’t limit yourself to computer-oriented solutions. Step back and try to get a larger view of the problem. Don’t worry if your analogy doesn’t match perfectly—the only reason for starting with an analogy is that it gives you a place to start. The best programmers are people who have broad experience in solving all kinds of problems.

3) Means-Ends Analysis

Often the beginning state and the ending state are given; the problem is to define a set of actions that can be used to get from one to the other. Suppose you want to go from Dutsinma to Kano, You know the beginning state (you are in Dutsinma) and the ending state (you want to be in Kano) The problem is how to get from one to the other. In this example, you have lots of choices. You can walk, ride a bike, drive a car. Join taxi or whatever. The method you choose depends on your circumstances. If you’re in a hurry, you’ll probably decide to fly. Once you’ve narrowed down the set of actions, you have to work out the details. It may help to establish intermediate goals that are easier to meet than the overall goal. You might decide to divide the trip into legs: campus to motor park and then park to kano Your intermediate goal is to get from Dutsinma to Kano. Now you only have to examine the means of meeting that intermediate goal. The overall strategy of means-ends analysis is to define the ends and then to analyze your means of getting between them. The process translates easily to computer programming. You begin by writing down what the input is and what the output should be. Then you consider the actions a computer can perform and choose a sequence of actions that can transform the input into the results.

4) Divide and Conquer

We often break up large problems into smaller units that are easier to handle. Cleaning the whole house may seem overwhelming; cleaning the rooms one at a time seems much more manageable. The same principle applies to programming. We break up a large problem into smaller pieces that we can solve individually.

The Building-Block Approach

Another way of attacking a large problem is to see if any solutions for smaller pieces of the problem exist. It may be possible to put some of these solutions together end to end to solve most of the big problem. This strategy is just a combination of the look-for familiar- things and divide-and-conquer approaches. You look at the big problem and see that it can be divided into smaller problems for which solutions already exist. Solving the big problem is just a matter of putting the existing solutions together, like mortaring together blocks to form a wall

Merging Solutions

Another way to combine existing solutions is to merge them on a step-by-step basis. For example, to compute the average of a list of values, we must both sum and count the values. If we already have separate solutions for summing values and for counting the number of values, we can combine them. But if we first do the summing and then do the counting, we have to read the list twice. We can save steps if we merge these two solutions: read a value and then add it to the running total and add 1 to our count before going on to the next value. Whenever the solutions to sub problems duplicate steps, think about merging them instead of joining them end to end.

Mental Blocks: The Fear of Starting

Writers are all too familiar with the experience of staring at a blank page, not knowing where to begin. Programmers have the same difficulty when they first tackle a big problem. They look at the problem and it seems overwhelming. Remember that you always have a place to begin solving any problem: Write it down on paper in your own words so that you understand it. Once you paraphrase the problem, you can focus on each of the subparts individually instead of trying to tackle the entire problem at once. This process gives you a clearer picture of the overall problem. It helps you see pieces of the problem that look familiar or that are analogous to other problems you have solved. And it pinpoints areas where something is unclear, where you need more information. As you write down a problem, you tend to group things together into small, understandable chunks of data and operations, which may be natural places to split the problem up—to divide and conquer. Your description of the problem may collect all of the information about data and results into one place for easy reference.

Rewriting the problem in your own words is a good way to focus on the subparts of the problem, one at a time, and to understand what is required for a solution.

Algorithmic Problem Solving

Coming up with an algorithm for solving a particular problem is not always cut-and dried. It fact, it is usually a trial-and-error process requiring several attempts and refinements. We test each attempt to see if it really solves the problem. If it does, fine. If it doesn’t, we try again. We typically use a combination of the techniques we’ve described to solve any nontrivial problem. Remember that the computer can only do certain things. Your primary concern, then, is how to make the computer transform, manipulate, calculate, or process the input data to produce the desired output. If you keep in mind the allowable instructions and data types in your programming language, you won’t design an algorithm that is difficult or impossible to code.

**INTRODUCTION TO BASIC PROGRAMMING**

In computer programming, **BASIC** (an acronym for **Beginner's All-purpose Symbolic Instruction Code** is a family of high-level programming languages.

The original BASIC was design in 1964, by John George Kemeny and Thomas Eugene Kurtz at Dartmouth College to provide access for non-science students to computers. At the time, nearly all use of computers required writing custom software, which was something only scientists and mathematicians can do. The language (in one variant or another) became widespread on microcomputers in the late 1970s and home computers in the 1980s. BASIC remains popular to this day in a handful of highly modified dialects and new languages based on BASIC such as Microsoft Visual Basic.

**The eight design principles of BASIC were:**

1. Easy for beginners to use.
2. A general-purpose programming language.
3. Allow advanced features to be added for experts (while keeping the language simple for beginners).
4. interactive.
5. Provide clear and friendly error messages.
6. Respond quickly for small programs.
7. Not to require an understanding of computer hardware.

**BASIC CHARACTER SETS**

While learning a natural language, we normally start from its alphabets which are called characters. These characters are broadly classified into three categories viz Alphabetic, Numeric and special characters

1. Alphabetic characters are the 26 alphabetic characters (A, B, C, … Z)
2. Numeric characters are the 10 numeric characters ( 0 1 2 3 4 5 6 7 8 9)
3. Special characters are (+ - \* / + = , . ( ) “ & $ ? etc )

**TYPES OF OPERATORS**

We have three classes of operators

1. Arithmetic operators
2. Relational operators
3. Logical operators

***Arithmetic operators:*** are those operators which permit arithmetic/mathematical operations/calculations to be performed on any given data (operand), they include **+, -, \*, /, \*\*, ^**

|  |  |
| --- | --- |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| / | Division |
| \*\* or ^ | Exponentiation |

***Relational operators:*** are those operators which allow for comparison. Below are some relational operators in BASIC and their meanings

|  |  |
| --- | --- |
| Operator | meaning |
| > | Greater than |
| < | Less than |
| = | Equal to |
| >= | Greater than or equal to |
| <= | Less than or equal to |
| < > | Not equal to |

**Logical operators**

These are operators which allow for a selection based on one or more conditions being true or false. These operators include OR, AND and NOT

Example: IF A = 1 OR B = 2 THEN PRINT “TRUE”

A < B AND A< C THEN PRINT “LAST = A”

**DATA TYPES**

BASIC programming language uses two types of data:-

1. **Numeric data types** eg 10, 20, 330.50, 2.5
2. **Alphanumeric/String** data eg “gina” “ P. o. Box 2041”

All data is represented in the computer in binary form. The type of data determines the way in which it is represented and the operations which may be performed on it. Data is usually referred to as variables.

In programming, a variable is a value (data) that can change, depending on conditions or on information passed to the program. Typically, a program consists of instructions that tell the computer what to do and data that the program uses when it is running. The data usually consists of ***constants*** or **fixed values** that never change and variable values (which are usually initialized to "0" or some default value because the actual values will be supplied by a program's user). Usually, both constants and variables are defined as certain data types.

**Reserved words** (occasionally called Keywords) are special words reserved by a programming language or by a program. You are not allowed to use reserved words as variable names. For example, in BASIC the words Print, Remark, Input, Locate, Data, etc are reserved because they have special meaning.

A **variable** is usually represented with a unique name which is called **variable name**. Program applies such a name to memory and uses it to refer to it. A variable name usually consists of alphabetic characters and decimal digits, beginning with an alphabetic character. It is usually good to give meaningful names to your variables. Examples: VOL, TEMP, A2, COLUMN, IBM370 etc. Names are given to memory locations for easy referencing. A variable name in BASIC consists of 1 to 40 characters. The string variable name always have dollar ($) sign attached to it as the last character.eg name$. It cannot be used in a formula during computation. The numeric variable can be used in computation, it does not have dollar sign ($) attached to it. E.g. Length, amount, price. It is advisable to give meaningful name to your variable. **Reserved words** should not be used in forming variable names.

#### Rules for forming variable names in Summary. 1. All variable names must start with a letter of the alphabet from A to Z. 2. No special symbols or basic key words must be used as variable name. 3. If more than one variable is used in a computer program, each variable name must be different. 4. Keep names short and meaningful as to their use.

#### 5. Variable names should help you remember what they hold. Good examples: age, salary, name, weight, etc. Bad examples: a, b, c, x, y, z. 6. The variable type or usage must be declared or specified by: a) use a name (with no special symbol) for a NUMERIC location. b) use a name with a $ sign at the end for an ALPHANUMERIC location.

**RULES FOR WRITING BASIC PROGRAMS**

1. It must start with a line number ( number statement)
2. Two statements can be combined in a line, but should be separated by comma
3. BASIC statements should always follow the number statement
4. Succeeding lines must be greater than the preceding lines
5. Numbers must be whole numbers
6. No repetition of number statement
7. There should be at least a space between the number statement and BASIC statement
8. A full stop is no allowed at the end of any statement the keyboard when the program
9. The program must always stop with an END statement.

**BASIC STATEMENTS**

1. The **REM** statement

REM: means remark to basic interpreter. It is used to indicate comment line. In a program. It can be replaced with a quote mark. Example:

10 Rem to sum three numbers

10 ‘To Sum Three Numbers

1. **INPUT** Statement

It is used to ask for input (data) through the keyboard when the program is running. It is used for interactive programming. INPUT Statement is followed by variable name, more than one variable name can declared with an input statement but should be separated by comma. When running the program the data should also be typed separated by comma as declared in the INPUT statement. E.g.

INPUT AMT, NAME$, discount

3 **THE READ AND DATA** Statement

This is another Statement used to ask for input (data), but this time the data is not expected from the keyboard but within the program. When the BASIC interpreter encounters a READ statement, it looks for the DATA statement which contains the actual data for the variables declared in the READ statement. The data statement must be used anytime the READ statement is used. The DATA statement must match with the READ statement variables declared.

e.g. READ NAME$, AMT

\_ \_

\_ \_

DATA ”Bag “, 30000

4 THE **PRINT** Statement

This is an output statement, it is used to tell the computer to return to us or show us the results from computation on the screen. Example: 10 PRINT A.

Where A is a variable name, more than one item may be listed in the print statements but they must be separated by commas or semicolons.

Example of BASIC programs using the above statements

1. A program to calculate the area of a room

Solution

10 REM A PROGRAM TO CALCULATE THE AREA OF A ROOM

20 INPUT “ENTER THE LENGTH”; LENGTH

30 INPUT “ENTER THE WIDTH”; WIDTH

40 LET AREA = LENGTH \* WIDTH

50 PRINT “ THE AREA OF THE ROOM IS”; AREA

60 END

1. To rewrite the program using the READ/DATA Statement

Solution

10 REM A PROGRAM TO CALCULATE THE AREA OF A ROOM

20 READ LENGTH, WIDTH

30 LET AREA = LENGTH \* WIDTH

40 DATA 100, 50

60 PRINT “ THE AREA OF THE ROOM IS”; AREA; “M2”

70 END

1. **LET** STATEMENT: The LET statement is used for assignments. The LET statement operates from right to left. That is to say, the value on the right of the equal sign is assigned to the variable on the left of the equal sign. The sign means an assignment and not equality. Example

let PI = 3.142, let sum = A + B + C

1. Program that accept three numbers (A, B, C), compare them and print the largest.

Solution

10 REM A PROGRAM THAT ACCEPTS THREE NUMBERS COMPARE THEM AND PRINT THE LARGEST

20 INPUT “ENTER THE NUMBERS “; A, B, C

30 IF A > B THEN BIGGER = A

40 ELSE BIGGER = B

50 IF BIGGER > C THEN BIGGEST = BIGGER

60 ELSE BIGGEST = C

70 PRINT BIGGEST

80 END

1. **GOTO** STATEMENT: is used to transfer execution to a specified line number. Example GOTO 20- this tells the computer to continue execution from line 20.
2. **IF – THEN – ELSE** STATEMENT

This statement is used to test whether a transfer of control or execution will take place or not. It is a powerful statement that gives the computer the ability to make decisions and take and alternative action based on the outcome. It is always used in conjunction with relational operators to set the condition that will be tested.

1. A program that will read in the marks scored by a student in six different courses and calculate the average score.

Solution

10 I = 0.0

20 SUM = 0.0

30 INPUT SCORE

40 I = I +1

50 SUM = SUM + SCORE

60 IF I <= 5 THEN GOTO 3O

70 AVERAGE = SUM/6

80 PRINT AVERAGE

90 END

NOTE: the ELSE statement is not mentioned in the above program, but line 70 is by default the implied ELSE statement because it is only executed when line 60 is no longer true.

1. **DIM** STATEMENT

This statement is used for declaration of array or matrix, it must be the first statement to be coded before any statement that reference that array or matrix. Array can be single dimensional or double dimensional.

Example:

DIM A(10)- This is a single dimensional array with 10 elements.

DIM B(5,4) – This is a double dimensional array or matrix named B with 5 rows and 4 columns given a total of 20 ( i.e. 5 \* 4) elements in the array.

1. A program that will read in 20 elements of an array called A and calculate their average.

Solution

10 DIM A(20)

20 FOR I = 1TO 20

30 INPUT A( I )

40 SUM = SUM + A(I)

50 NEXT I

60 AVERAGE = SUM/20

70 PRINT AVERAGE

80 END

1. A program to read and print the elements of a 3 x 4 matrix.

Solution

10 DIM A (3,4)

20 I = I + 1

30 J = J + 1

40 INPUT A(I,J)

50 PRINT A(I,J)

60 IF J <= 4 THEN GOTO 30

70 J = 0

80 IF I <=2 THEN GOTO 20

90 END

1. **CLS** STATEMENT

This is a statement used to clear or tidy up the screen. Example:

10 CLS –This clears the previous content of the screen or display.

**EXPRESSIONS**

**EXPRESSION**: an expression is a rule which when evaluated will give a single value, in its simplest form it consist of two operands with an operator between them. In basic there are numeric expression, logical expression, and string expression.

**NUMERIC EXPRESSION**:- to understand numeric expression let us look at arithmetic operators, arithmetic operators are used to indicate arithmetic operation such as addition, subtraction, division, multiplication and exponentiation.

**ARITHMETIC OPERATORS**

Addition: + (Plus sign)

Subtraction: - (Minus sign )

Division: / (Slash)

Exponentiation: ^ (Caret , or Upward-pointing arrow)

Therefore, these arithmetic operators are used to connect numeric constants and numeric variables to form ***Numeric expression****.*

**Examples of numeric expressions are:**

A + B + C - D A + B ( 5 \* P – 2 \* Q ) / ( X + Y )

COUNT + 1 3.142 \* R ^ 2 4 \* PI \* RADIUS ^ 3 / 3

“Many versions of BASIC include two additional arithmetic operators: integer division (\) and integer remainder (MOD). In integer division, each of the two given numbers is first rounded to an integer; the division is then carried out on the rounded values and the resulting quotient is truncated to an integer. The integer remainder operation provides the remainder resulting from an integer division”.

**Examples**

13 / 5 = 2.6 13 \ 5 = 2 13 MOD 5 = 3

8.6 / 2.7 = 3.185185 8.6 \ 2.7 = 3 8.6 MOD 2.7 =0

8.3 / 2.7 = 3.074074 8.3 \ 2.7 =2 8.3 MOD 2.7 = 2

**HIERACHY OF OPERATIONS**

There are some situations when several operators appear in an expression, to solve these problems we therefore apply hierarchy of operations rule. The hierarchy of operations is

1. Exponentiation: all exponentiation operations are performed first.
2. Multiplication and division: they have the same hierarchy, it means any one that comes first will be evaluated before the other one, but they are performed after exponentiation.
3. Integer division: in those version of BASIC that include this operation, integer division operations are carried out after all multiplication and (ordinary) division operations.
4. Integer remainder: in those version of BASIC that include this operation, integer remainder operations are carried out after all integer divisions operations.
5. Addition and subtraction:-they have the same hierarchy but all the above operations must be performed first.