**Chapter 1: Introduction to Computers**

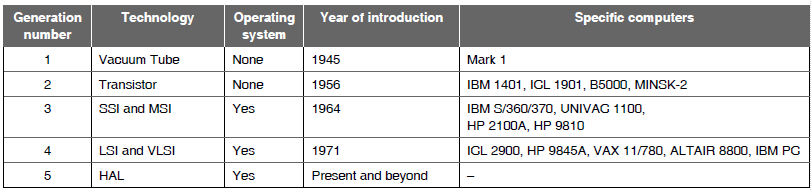
**1.1 Introduction**

A computer is a programmable machine that receives data, stores and manipulates data, and provides output in a useful format.

In regards to today’s computers, the “machine” part of the computer is called the hardware, while the “programmable” part is called the software.”

BINARY is the data format that the computer understands: not a high-level or low-level programming language or a human language. All other data should be reduced and converted to this BINARY format so that a computer system can understand and work on it.

Generation of computers

****

**Organization of a computer**

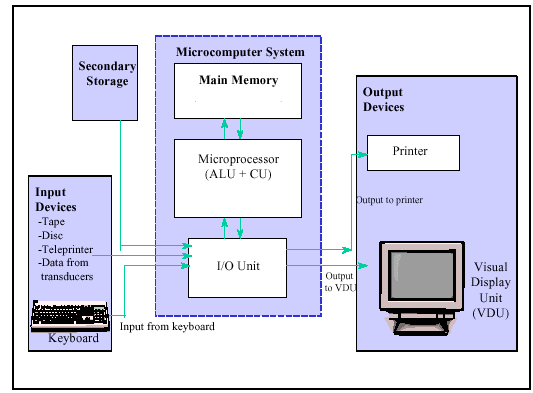


Figure 1: Elements of a computer

The main elements associated with a computer system are as follows:

**1. Central Processing Unit (CPU)/Microprocessor** – CPU performs the computation of data and controlling the hardware and Input/output devices.

A central processing unit (CPU) is the electronic circuitry within a computer that carries out the instructions of a computer program by performing the basic arithmetic, logical, control and input/output (I/O) operations specified by the instructions.

Arithmetic Logic Unit (ALU) is digital circuitry for arithmetic, logical and comparison operations (addition, subtraction, multiplication, division, AND, OR etc.).

The Control Unit (CU) is digital circuitry contained within the processor that coordinates the sequence of data movements into, out of, and between a processor's many sub-units. It fetches instructions from memory, decodes the instruction and executes the instruction.

**2. Main Memory (primary storage)** – Holds the data that is fetched from secondary storage which would be needed for computation. It is a volatile memory; the data will be wiped off when the computer the power supply is disconnected. Example of primary storage is a RAM (Random Access Memory)

**3. Secondary Storage** – Stores the data permanently. Its non-volatile memory and stores the data even when powered off. Example for secondary storage devices would be ROM, Magnetic tapes, Hard Drives etc.

**4. Input and Output (I/O) devices** – The devices that help in obtaining input from different devices such as keyboard or tapes and display the output to a Monitor or a printer.

The two components at the heart of the hardware structure are the central processing unit (CPU) and the main Memory. The CPU monitors and controls the operation of the other devices and the flow of information to and from these devices, and it performs all the necessary manipulations of the data. The Main Memory is used to store information for immediate access by the CPU. Main Memory is also referred to as Primary Storage or Main Store. Secondary storage devices provide permanent storage of large amounts of data. Secondary storage is also called: secondary memory, external memory, backing store or auxiliary storage. This storage may consist of magnetic tapes, magnetic disk, optical memory device, or similar device.

Input/output devices provide an interface between the computer and the user. There is at least one input device (e.g. keyboard, mouse, measuring device such as a temperature sensor) and at least one output device (e.g. printer, screen, control device such as an actuator). Input and output devices like keyboards and printers, together with the external storage devices, are referred to as peripherals

The computer system requires interconnections between the various components. When these data paths carry more than one bit simultaneously from a number of different components, it is referred to as a data bus or simply bus.

**1.2 Storage Devices**

Storage devices are one of the core components of any computing device. They store all the data and applications on a computer. They are available in different form factors depending on the type of underlying device. For example, a standard computer has multiple storage devices including RAM, cache, a hard disk, an optical disk drive and externally connected USB drives

Storage device can be classified as Primary and Secondary memory devices which are explained in the upcoming sections.

**1.2.1 Primary storage devices**

* Primary memory is also known as main memory or internal memory. Primary memory is the memory that is directly accessed by the CPU to store and retrieve information. RAM (Random Access Memory) and ROM (Read Only Memory) serve as primary memory units.
* Primary memory holds only those data and instructions on which computer is currently working.
* It is a volatile memory when we speak about RAM (Data is lost in case power is switched off); However, ROM is non-volatile primary memory and retains the data even when the power supply is disconnected.
* Primary memory devices are generally made up of semiconductors.
* Primary memory devices are faster than secondary memory devices in terms of read and write speeds.

**1.2.2 Secondary Storage Devices**

* Secondary memory is also known as external memory.
* Secondary memories are used for storing data/Information permanently.
* Secondary memory units are non-volatile. They retain information even when the power supply is disconnected
* For CPU to access the contents present in secondary memory devices, the contents of secondary memories are first transferred to main memory, and then accessed by CPU.
* Examples of secondary storage devices can be hard disk drives, CD and DVD ROMs, Flash drives etc.

**1.2.3 Differences between Primary and Secondary Memory**

|  |  |  |
| --- | --- | --- |
| **Serial No.** | **Primary Memory** | **Secondary Memory** |
| 1 | It can be volatile or non-volatile memory. | It is usually non-volatile memory. |
| 2 | Access time is faster than secondary memories | Access time is slower than  primary memories |
| 3 | Primary memories are usually connected via slots | Secondary memories are connected via ports or cables. |
| 4 | Price of the primary memories is higher as they are faster and compact. | Price of the secondary memory devices are high as they are bulky and slow when compared to primary memory devices. |
| 5 | At present the capacities of primary memory ranges from 512 MB to 16GB for desktop computers and can be of higher capacities for Servers. | The capacities of hard disk drive currently can be up to 60 Tera Bytes. |
| 6 | The contents of the primary memory are directly accessible to the CPU. | The contents of the secondary memory are not directly accessible by the CPU. The contents have to be first copied to the primary memory and then be accessed by the CPU. |
| 7 | This is a temporary memory when we speak in terms of RAM. | Secondary memory devices can be used to store data permanently |
| 8 | These are made up of semiconductor memories. | These are made up of magnetic  and optical memories. |
| 9 | Example: RAM and ROM. | Example: DVD ROM, CD-ROM and  Hard disk. |

**1.2.4 Random Access Memory (RAM)**

* RAM is an internal memory which stores frequently used program instructions and data to increase the general speed of a system. A RAM allows data items to be read or written in almost the same amount of time irrespective of the physical location of data inside the memory.
* RAM holds the data as long as there is power supply. If the power supply is disconnected or the system is shut down, the data which was stored in the RAM is lost. Hence RAM is a volatile memory device. An uninterrupted power supply would be required to ensure the data is not lost.
* RAM memory units are available as Integrated Circuits and are expensive.
* RAM is small, both in terms of its physical size and in the amount of data it can hold.

*Types of RAM:*

* Static RAM (SRAM)
* Dynamic RAM (DRAM)

*Static RAM*

* The word static indicates that the memory retains its contents as long as power is being supplied.
* The data is lost when the power supply is disconnected due to volatile nature.
* SRAM cell is constructed with a matrix of 6-transistors and but no capacitors are used to form a SRAM cell.
* There is no leakage of current/charge in SRAM cell, so SRAM cell does not require a refresh circuit to retain charge.
* Due to the presence of 6 transistors for making up on SRAM cell, SRAM uses more space than DRAM for the same amount of storage space, thus making the manufacturing costs higher. It takes more transistors to store a bit of data using SRAM.
* SRAM is used as cache memory and can be accessed in a very short of time.

*Characteristics of SRAM*

* Reliability
* Simplicity: There is no need of a refresh circuit.
* Faster
* Used as cache memory
* Large size
* Expensive
* Lower power consumption

*Dynamic RAM*

* DRAMs are made up of memory cells which are composed of one transistor and one capacitor.
* The capacitor keeps discharging with time and hence a DRAM cell must be continually refreshed in order to maintain the data.
* Refreshing of DRAM cell is done with the help of refresh circuit that rewrites the data present on the RAM periodically.
* DRAM is used for most of the applications as system RAM memory as it is comparatively cheaper and smaller when compared to SRAM.

*Characteristics of the Dynamic RAM:*

* It has short data lifetime.
* Need to be refreshed continuously.
* Slower access speeds when compared to SRAM.
* Used as RAM in computer systems.
* Smaller in size.
* Less expensive.
* Higher power consumption when compared to SRAM.

*Differences between SRAM and DRAM*

|  |  |  |
| --- | --- | --- |
| **Serial No.** | **Static RAM (SRAM)** | **Dynamic RAM (DRAM)** |
| 1 | It is faster than DRAM. | It is slower than SRAM. |
| 2 | It is more expensive as compared to DRAM. | It is less expensive. |
| 3 | An SRAM cell is made up of 6 transistors | A DRAM cell is made up of 1 transistor and a capacitor |
| 4 | It does not need to be refreshed. | It has to be refreshed after each read operation. |
| 5 | It consumes less power as there is no refresh circuitry. | It consumes more power. |
| 6 | It holds data indefinitely as long as the computer is turned on. | It holds data dynamically not indefinitely. |
| 7 | SRAM is used as cache memory because it has very fast access. | DRAM is used for most system memory because it is cheap and small. |
| 8 | SRAM uses more transistors per bit of memory | DRAM uses 1 transistor per bit of memory |

**1.2.5 Read Only Memory (ROM)**

* ROM stands for Read Only Memory. (It is also called as firmware memory).
* As the name says “read only” we can only read the content of it and not write any information to it. Different types of ROMs are available to which information can be written to and these will be discussed in the further sections.
* ROM is a non-volatile type of memory. The data written to it remains unaltered even if the power is disconnected.
* The information is stored permanently in ROM during manufacturing time in case of One Time Programmable ROMs.
* ROMs are used to store instructions that are required to start a computer (boot sequence). For such applications, the ROM is used as bootstrap memory.
* ROMs are not only used in the computer but also in our day to day lives in electronic appliances such as washing machines and microwave oven.

*Types of ROMs*

* PROM (Programmable Read Only Memory)
* EPROM (Erasable and Programmable Read Only Memory)
* EEPROM (Electrically Erasable and Programmable Read Only Memory)

*Programmable Read Only Memory (PROM)*

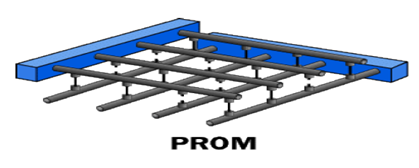


Fig 1.8: Programmable Read Only Memory (PROM)

* PROM is programmable read-only memory that can be modified only once by a user. It is also called as One Time Programmable ROM (OTP ROM).
* The user buys a blank PROM and enters the desired contents using a PROM program.
* A PROM memory array consists of an array of fuses which are burnt open (to write a logic 0) during programming. Initially an unprogrammed PROM contains all 1’s (all the fuses form a closed circuit) as shown in fig 1.8.
* A PROM can be programmed only once and is not erasable.
* The process of writing data to the PROM involves a special piece of equipment called a device programmer.
* The device programmer writes data to the device one word at a time by applying an electrical charge to the input pins of the PROM chip.
* Once a PROM has been programmed, its contents can never be changed.

*Erasable Programmable ROM (EPROM)*

**

Fig 1.9: Erasable Programmable ROM with a window to pass UV light to erase the contents

* An EPROM (erasable-and-programmable ROM) is programmed in exactly the same manner as a PROM. However, EPROMs can be erased and reprogrammed repeatedly.
* To erase an EPROM, we simply expose the device to a strong source of ultraviolet light. (A window in the top of the device allows the light to reach the silicon.) as shown in the figure 1.9.
* By doing this, the chip is essentially reset to its initial-unprogrammed-state.
* Though more expensive than PROMs, their ability to be reprogrammed makes EPROMs an essential part of the software development and testing process.
* EPROMs were used in memory sticks, smart memories which were later replaced by EEPROM (Electrically Erasable PROM).

*Electrically Erasable Programmable ROM (EEPROM)*

* EEPROM is an electrically-erasable-and-programmable ROM.
* Internally, they are similar to EPROMs, but the erase operation is accomplished electrically, rather than by exposure to ultraviolet light.
* Any byte within an EEPROM may be erased and rewritten.
* Once written, the new data will remain in the device forever-or at least until it is electrically erased.
* The primary tradeoff for this improved functionality is higher cost.
* It is used in RTCs (Real Time Clock), digital potentiometers, digital temperature sensors, video games cartridges, computer BIOS that were built after 1984.

**1.3 Classification of Computers**

There are four classifications of digital computer systems: super-computer, mainframe computer, minicomputer, and microcomputer.

**Super-computers** are very fast and powerful machines. Their internal architecture enables them to run at the speed of tens of MIPS (Million Instructions per Second). Super-computers are very expensive and for this reason are generally not used for CAD applications. Examples of supercomputers are: Cray and CDC Cyber 205.

**Mainframe computers** are built for general computing, directly serving the needs of business and engineering. Although these computing systems are a step below super-computers, they are still very fast and will process information at about 10 MIPS. Mainframe computing systems are located in a centralized computing center with 20-100+ workstations. This type of computer is still very expensive and is not readily found in architectural/interior design offices.

**Minicomputers** were developed in the 1960's resulting from advances in microchip technology. Smaller and less expensive than mainframe computers, minicomputers run at several MIPS and can support 5-20 users. CAD usage throughout the 1960's used minicomputers due to their low cost and high performance. Examples of minicomputers are: DEC PDP, VAX 11.

**Microcomputers** were invented in the 1970's and were generally used for home computing and dedicated data processing workstations. Advances in technology have improved microcomputer capabilities, resulting in the explosive growth of personal computers in industry. In the 1980's many medium and small design firms were finally introduced to CAD as a direct result of the low cost and availability of microcomputers. Examples are: IBM, Compaq, Dell, Gateway, and Apple Macintosh.

The average computer user today uses a microcomputer. These types of computers include PC's, laptops, notebooks, and hand-held computers such as Palm Pilots.

**1.4 Number Systems**

A number system (or **numeral system)** is the one which is used for expressing (or) writing numbers. It uses digits or other symbols to represent the numbers of a given set in appropriate manner. Commonly used number systems are Decimal, Binary, Octal and Hexadecimal systems. Each system has its own way of representing the numbers. For ex: 101 can be seen as “one hundred one” in decimal notation, “five” in binary system and “sixty five” in octal system.

**Table: description of number systems**

|  |  |
| --- | --- |
| **S.No** | **Description of Number System** |
| 1 | **Decimal Number System** Base 10. Digits used : 0 to 9 |
| 2 | **Binary Number System** Base 2. and Bits used : 0, 1 |
| 3 | **Octal Number System** Base 8. Digits used : 0 to 7 |
| 4 | **Hexadecimal Number System** Base 16. Digits used: 0 to 9, Letters used : A- F |

**Decimal Number System**

The number system that we use in our day-to-day life is the decimal number system. Decimal number system has base 10 as it uses 10 digits from 0 to 9. In decimal number system, the successive positions to the left of the decimal point represent units, tens, hundreds, thousands, and so on.

Characteristics of the Decimal number system are as follows

* It uses ten digits which are from 0 to 9.
* This is also called as base 10 number system.
* Each position in a decimal number represents a **(x)** power of the base (10). Example **10x**
* Last position in a decimal number represents a **(y)** power of the base (10). Example **10y** where **y** represents the last position-1.

**Example 1: The decimal Number (4215)10 is represented** as

|  |  |  |
| --- | --- | --- |
| Step | Binary Number | Decimal Number |
| Step 1 | (4215)10 | ((4 x 103) + (2 x102) + (1 x 101) + (5 x 100))10 |
| Step 2 |  | (4000 +200 + 10 + 5)10 |
| **Ans = (4215)10** | | |

**Example 2: The decimal number (536.159)10 is can be written as**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Integer part | | |  | Fractional Part | | |
| 102(MSB) | 101 | 100 | **.** | 10**-1** | 10**-2** | 10-3(LSB) |
| 5 | 3 | 6 | Binary point | 1 | 5 | 9 |
| 5x102 | 3x101 | 6x100 | 1x10-1 | 5x10-2 | 9x10-3 |
| **Ans= 500+30+6+0.1+0.05+0.009 = (536.159)10** | | | | | | |

**Binary Number System**

Characteristics of the binary number system are as follows −

* Uses two bits, 0 and 1
* Also called as base 2 number system
* Each position in a binary number represents a **(x)** power of the base (2). Example **2x**
* Last position in a binary number represents a **(y)** power of the base (2). Example **2y** where y represents the last position-1.

**Example 3: Represent / Convert binary Number (10101)2 in decimal system**

|  |  |  |
| --- | --- | --- |
| **Step** | **Binary Number** | **Decimal Number** |
| Step 1 | (10101)2 | ((1 x 24) + (0 x 23) + (1 x 22) + (0 x 21) + (1 x 20))10 |
| Step 2 |  | (16 + 0 + 4 + 0 + 1)10 |
| **Ans = (21)10** | | |

**Example 4: Represent / Convert binary Number (1011.011)2 in decimal system**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Integer part | | | |  | Fractional Part | | |
| 23(MSB) | 22 | 21 | 20 | **.** | 2**-1** | 2**-2** | 2-3(LSB) |
| 1 | 0 | 1 | 1 | Binary point | 0 | 1 | 1 |
| 1x23 | 0x22 | 1x21 | 1x20 | 0x2-1 | 1x2-2 | 1x2-3 |
| **Ans = 8+0+2+1+0+0.25+0.125 = (11.375)10** | | | | | | | |

## Octal Number System

Characteristics of the octal number system are as follows

* Uses eight digits, 0,1,2,3,4,5,6,7
* Also called as base 8 number system
* Each position in an octal number represents a **(x)** power of the base (8). Example **8x**
* Last position in an octal number represents a **(y)** power of the base (8). Example **8y** where (**y)** represents the last position-1

### Example 5: Represent octal Number (12570)8 in decimal form

|  |  |  |
| --- | --- | --- |
| **Step** | **Octal Number** | **Decimal Number** |
| Step 1 | (12570)8 | ((1 x 84) + (2 x 83) + (5 x 82) + (7 x 81) + (0 x 80))10 |
| Step 2 |  | (4096 + 1024 + 320 + 56 + 0)10 |
| **Ans = (5496)10** | | |

### Example 6: Represent octal Number (231.25)8 in decimal form as

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Integer part** | | |  | **Fractional Part** | |
| 82 | 81 | 80 | **.** | 8**-1** | 8**-2** |
| 2 | 3 | 1 | Decimal point | 2 | 5 |
| 2x82 | 3x81 | 1x80 | 2x8-1 | 5x8-2 |
| **Ans = 128+24+1+0.25+0.078125 = (153.328125)10** | | | | | |

## Hexadecimal Number System

Characteristics of hexadecimal number system are as follows

* Uses 10 digits and 6 letters, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
* Letters represent the numbers starting from 10. A = 10. B = 11, C = 12, D = 13, E = 14, F = 15
* Also called as base 16 number system
* Each position in a hexadecimal number represents a **(x)** power of the base (16). Example, **16x**
* Last position in a hexadecimal number represents a **(y)** power of the base (16). Example **16y** where **(y)** represents the last position-1.

### Example 7: Represent Hexadecimal Number (A124)16 in decimal form

|  |  |  |
| --- | --- | --- |
| **Step** | **Hexadecimal Number** | **Decimal Number** |
| Step 1 | (A124)8 | ( (1 x 163) + (1 x 162) + (2 x 161) + (4 x 160))10 |
| Step 2 |  | ( 4096 + 256 + 32 + 4)10 |
| **Ans = (4388)10** | | |

### Example 8: Represent Hexadecimal Number (A02.B7)16 in decimal form

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Integer part** | | |  | **Fractional Part** | |
| 162 | 161 | 160 | **.** | 16**-1** | 16**-2** |
| A | 0 | 2 | Decimal point | B | 7 |
| 10x162 | 0x161 | 2x160 | 11x16-1 | 7x16-2 |
| **Ans = 2560+0+2+0.6875+0.02734375 = (2562.71484)10** | | | | | |

**Applications of number system:**

* Decimal system is widely used in mathematical notation.
* Binary system is used in DIGITAL signal coding of binary, as it uses only two digits one and zero to form different digits. It is also used in Nim like games and modern technology of electronic digital computers.
* Octal became widely used in computing systems such as the PDP-8, ICL 1900 and IBM mainframes employed 12-bit, 24-bit or 36-bit words. It also cuts cost by allowing Nixie tubes, seven-segment displays, and calculators to be used for the operator consoles, where binary displays were too complex to use. They are also used in some programming languages (C, Perl, PostScript…) for textual/graphical representations of byte strings.
* The **hexadecimal** system is commonly used by **programmers** to describe locations in memory because it can represent every byte (i.e., eight bits) as two consecutive **hexadecimal** digits instead of the eight digits that would be required by binary (i.e., base 2) numbers and the three digits that would be required with decimal. It is also used to describe colors on web pages.

The below table below shows the Binary counting sequence and the representation to different numbers systems.

**Table: The representation of numbers in different number systems**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Binary | |  | Decimal | Octal | Hexadecimal |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 2 | 2 | 2 |
| 0 | 0 | 1 | 1 | 3 | 3 | 3 |
| 0 | 1 | 0 | 0 | 4 | 4 | 4 |
| 0 | 1 | 0 | 1 | 5 | 5 | 5 |
| 0 | 1 | 1 | 0 | 6 | 6 | 6 |
| 0 | 1 | 1 | 1 | 7 | 7 | 7 |
| 1 | 0 | 0 | 0 | 8 |  | 8 |
| 1 | 0 | 0 | 1 | 9 |  | 9 |
| 1 | 0 | 1 | 0 | 10 |  | A |
| 1 | 0 | 1 | 1 | 11 |  | B |
| 1 | 1 | 0 | 0 | 12 |  | C |
| 1 | 1 | 0 | 1 | 13 |  | D |
| 1 | 1 | 1 | 0 | 14 |  | E |
| 1 | 1 | 1 | 1 | 15 |  | F |

**Number System Conversion**

Types of Number system conversion:

1. Binary to Octal and Hexadecimal Conversion
   * Binary to octal
   * Binary to hexadecimal
2. Octal and Hexadecimal Conversion to Binary
   * Octal to binary
   * Octal to Hexadecimal
   * Hexadecimal to Binary
   * Hexadecimal to Octal
3. Decimal to Any Radix
   * Decimal to binary
   * Decimal to octal
   * Decimal to hexadecimal
4. Any radix to Decimal
   * Binary to decimal
   * Octal to decimal
   * Hexadecimal to decimal

## Binary to Octal and Hexadecimal Conversion

**Example 9: Convert the binary number (001001111.110101010)2 to octal**

Solution:

1. Starting from the radix point partition the given binary number into **groups of three**, as per the given directions (left and right).

(001, 001, 111 **.** 110, 101, 010)2

1. Each group of three corresponds to a single octal digit. Using table 2.2 find the octal number for each group.

(001, 001, 111 **.** 110, 101, 010)2 = **(117.652)8**

**Example 10: Convert the binary number (1011001100.00110)2 to octal**

Solution:

1. Partition the given binary number into **groups of three**, starting at the radix point and going left and right.

(1,011, 001, 100 **.** 001, 10)2

1. Each group of three corresponds to a single octal digit. But the last group on both left hand and right hand side doesn’t have three digits. So add sufficient zero’s in the same direction to make a group of three digits.

(001, 011, 001, 100 **.** 001, 100)2 = **(1314.14)8**

**Example 11: Convert the binary number (111111001010.01111111)2 to hexadecimal**

Solution:

1. Partition the given binary number into **groups of four**, starting at the radix point and going left and right and right.

1111, 1100, 1010 **.** 0111, 11112

1. Each group of four corresponds to a single octal digit. Using table 4.3 find the octal number for each group.

1111, 1100, 1010 **.** 0111, 11112  = **(FCA.7F)16**

**Example 12: Convert the binary number 1111101010.01111102 to hexadecimal**

1. Partition the given binary number into **groups of four**, starting at the radix point and going left and right.

(11, 1110, 1010 **.** 0111, 110)2

1. Each group of four corresponds to a single hexadecimal digit. But the last group on both left hand and right hand side doesn’t have four digits. So add sufficient zero’s in the same direction to make a group of four digits.

(0011, 1110, 1010 **.** 0111, 1100)2 = **(3EA.7C)16**

**Octal and Hexadecimal Conversion to Binary**

**Example 13: Convert the octal number 735.58 to binary**

Solution: Each octal digit represents three binary bits. Use table 2.2 to find the three bits that correspond to each octal digit and replace it with the binary digits.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 7 | 3 | 5 | **.** | 5 |
| 111 | 011 | 101 | **.** | 101 |
| **Ans = 735.58 = (111011101.101)2** | | | | |

**Example 14: Convert the Hexadecimal number 2DE5.6A16 to binary**

Solution: Each hexadecimal digit represents four binary digits. Use table 2.2 to find the four bits that correspond to each hexadecimal digit and replace it with the binary digits.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | D=13 | E=14 | 5 | **.** | 6 | A=10 |
| 0010 | 1101 | 1110 | 0101 | **.** | 0110 | 1010 |
| **Ans = 2DE5.6A16 = (00101101 1110 0101.0110 1010)2** | | | | | | |

**Example 15: Convert the octal number (572.15)8 to hexadecimal**

**Step 1:** In order to convert octal number to hexadecimal number, first the octal number is to be converted into binary.

**Step 2:** Partition the above binary number into a group of four bits and write equivalent hexadecimal number.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 5 | 7 | 2 | **.** | 1 | 5 |
| 101 | 111 | 010 | **.** | 001 | 101 |
| **Step 1 = (572.15)8 = (101111010.001101)2** | | | | | |
| 101111010**.**001101 | | | | | |
| **Step 2:** 0001,0111,1010 **.** 0011,0101 | | | | | |
| **Final Ans: (17A.35)16** | | | | | |

**Example 16: Convert the Hexadecimal number 1BE5.3A16 to octal**

**Step 1:** In order to convert hexadecimal number to octal number first the hexadecimal number is to be converted into binary.

**Step 2:** Partition the above binary number into a group of three bits and write equivalent octal number.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | B | E | 5 | **.** | 3 | A |
| 0001 | 1011 | 1110 | 0101 | **.** | 0011 | 1010 |
| **Step 1:** (1BE5.3A)16 = (0001101111100101.00111010)2 | | | | | | |
| 0001101111100101**.**00111010 | | | | | | |
| **Step 2:** 000,001,101,111,100,101 **.** 001,110,010 | | | | | | |
| **Final Ans: (15745.162)8** | | | | | | |

## Any base/radix to Decimal conversion

**Binary to Decimal Conversion**

Refer to Example 2.3. & 2.4

**Octal to Decimal conversion**

Refer to Example 2.5 & 2.6

**Hexadecimal to Decimal conversion**

Refer to Example 2.7 & 2.8

**Decimal to Any radix/base conversion**

**Decimal to binary conversion**

**Example 17: Convert 2510 to binary**

Solution:

|  |  |  |
| --- | --- | --- |
| **Division** | **Remainder** | **Binary** |
| 25/2 | = 12+ remainder of 1 | 1 (Least Significant Bit) |
| 12/2 | = 6 + remainder of 0 | 0 |
| 6/2 | = 3 + remainder of 0 | 0 |
| 3/2 | = 1 + remainder of 1 | 1 |
| 1/2 | = 0 + remainder of 1 | 1 (Most Significant Bit) |
| **Ans: (25)10 = (11001)2** | | |

**Fractions Conversion**

In order to convert binary fraction into its equivalent decimal number each bit in the fractional part is multiplied by its positional weight after the decimal point and representing the values from the MSB to the LSB.

**Example 18:** **Convert 0.7510 to binary**

Solution: Fraction

Multiply .75 by 2 = 1.5  1 (MSB)

Multiply .5 by 2 = 1.0  1

Multiply 0 by 2 = 0.0  0 (LSB)

**Ans: (.75)10 = (.110)2**

**Example 19: Convert 95.062510 to binary**

Solution: The conversion is accomplished in two parts. First, convert the integer part of the decimal number by successive division.

|  |  |  |
| --- | --- | --- |
| **Division** | **Remainder** | **Binary** |
| 95/2 | = 47+ remainder of 1 | 1 (Least Significant Bit) |
| 47/2 | = 23 + remainder of 1 | 1 |
| 23/2 | = 11 + remainder of 1 | 1 |
| 11/2 | = 5 + remainder of 1 | 1 |
| 5/2 | = 2 + remainder of 1 | 1 |
| 2/2 | = 1 + remainder of 0 | 0 |
| 1/2 | = 0 + remainder of 1 | 1 (Most Significant Bit) |
| **Ans: (95)10 = (1011111)2** | | |

**Second, convert the fraction.**

Multiply .0625 by 2 = 0.125  0 (MSB)

Multiply .125 by 2 = 0.25  0

Multiply .25 by 2 = 0.5  0

Multiply .5 by 2 = 1.0  1

Multiply 0 by 2 = 0.0  0 (LSB)

**Ans: (.0625)10 = (.0001)2**

**Therefore the final Ans**: **(95.0625)10 = (1011111.0001)2**

**Example 20: Convert 177.62510 to Octal**

Solution:

1. Convert the integer portion by successive division

|  |  |  |
| --- | --- | --- |
| **Division** | **Result** | **Octal** |
| 177/8 | = 22+ remainder of 1 | 1 (Least Significant Digit) |
| 22/ 8 | = 2 + remainder of 6 | 6 |
| 2 / 8 | = 0 + remainder of 2 | 2 (Most Significant Digit) |
| **Ans: (177)10 = (261)8** | | |

Convert the fractional portion by successive multiplication

8 x .625 = 5.00  5 (MSD)

**Therefore the final Ans:**  **(177.625)10 = (261.5)8**

**Example 21: Convert (378)10 to hexadecimal**

Solution:

|  |  |  |  |
| --- | --- | --- | --- |
| **Division** | **Result** | **Hexadecimal** |  |
| 378/16 | = 23+ remainder of 10 | A (LSD) |
| 23/16 | = 1 + remainder of 7 | 7 |
| 1/16 | = 0 + remainder of 1 | 1 (MSD) |
| **Ans: 37810  = 17A16** | | | |

**Example 22: Convert (253.785)10 to hexadecimal.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Division** | **Result** | **Hexadecimal** |  |
| 253/16 | = 15+ remainder of 13 | D (LSD) |
| 15/16 | = 0 + remainder of 15 | F |
| **Ans: (253)10  = FD16** | | | |

Convert the fractional portion by successive multiplication

16 x 0.785 = 12.56  (12) C (MSB)

16 x 0.56 = 8.96  8

16 x 0.96 = 15.36  (15) F (LSB)

**Therefore the final Ans:**  **(253.785)10 = (FD.C8F)16**

**Exercise Problems:**

|  |  |
| --- | --- |
| 1) Convert (2026)8 to decimal  =2x83 + 0x82 + 2x81 + 6x80  = 1024 + 0 +16 + 6 = (1046)10 | 2) Convert (1ABCD)16 to decimal  =1x164 + Ax163 + Bx162 + Cx161 + Dx160  = (109517)10 |
| 3) Convert Binary to Decimal  (1110.111)2=( )10  =1x23 +1x22+ 1x21+ 0x20 +1x2-1+1x2-2+1x2-3  =8+4+2+0+0.5+0.25+0.125  =(14.875)10 | 4)Convert Binary to Decimal  (0.110)2= ( )10  = 1x2-1 + 1x2-2 + 0x2-3  =0.5+0.25+0  =0.75=( 0.75 )10 |
| 5) Decimal to Binary  (105.202)10 =(1101001.0011)2   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 105 |  |  | 0.202x2 | 0.404 | | 2 | 52 | 1 |  | .404x2 | 0.808 | | 2 | 26 | 0 |  | 0.808x2 | 1.616 | | 2 | 13 | 0 |  | 0.616x2 | 1.232 | | 2 | 6 | 1 |  |  | | | 2 | 3 | 0 |  | |  | 1 | 1 |  | | 6) Decimal to Binary  (0.825)10 = (0.1101)2   |  |  |  |  | | --- | --- | --- | --- | | 0.825x2 | = 1.650 | Carry | 1 | | 0.650x2 | = 1.300 |  | 1 | | 0.300x2 | = 0.600 |  | 0 | | 0.600x2 | = 1.200 |  | 1 | |
| 7) Binary to Octal  (0.101011)2 = (0.53)8   |  |  | | --- | --- | | 0.101 | 011 | | 5 | 3 | | 8)Octal to Binary  (3565)8 =(11101110101)2   |  |  |  |  | | --- | --- | --- | --- | | 3 | 5 | 6 | 5 | | 011 | 101 | 110 | 101 | |
| 9)Binary to Hex  (10110101001.101011)2=(5A9.AC)16   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 0101 | 1010 | 1001 | **.** | 1010 | 1100 | | 5 | A | 9 |  | A | C | | 10) Hex to Binary  (90AB.1C)16=(1001000010101011.00011100)2   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | 9 | 0 | A | B | **.** | 1 | C | | 1001 | 0000 | 1010 | 1011 |  | 0001 | 1100 | |
| 11) Decimal to Octal  (985.85)10=(1731.66314)8   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | 8 | 985 |  |  | 0.85x8= | 6.80 | C | 6 | | 8 | 123 | 1 |  | 0.80x8= | 6.40 |  | 6 | | 8 | 15 | 3 |  | 0.40x8= | 3.20 |  | 3 | |  | 1 | 7 |  | 0.20x8=  0.60x8= | 1.60  4.80 |  | 1  4 | | 12) Octal to Decimal  (1234.56)8=   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | 3 | 2 | 1 | 0 | . | -1 | -2 | | 1 | 2 | 3 | 4 | . | 5 | 6 |   =1x83+ 2x82+ 3x81+ 4x80+ 5x8-1+ 6x8-2  = 512+ 128 + 24 +4+ 0.625 +.09375  =(668.71875)10 |
| 13) Decimal to Hexadecimal  (988.86)10 =(?)16   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | 16 | 988 |  |  | 0.86x16 | =13.76 | (D)16 | | 16 | 61 | C |  | 0.76x16 | =12.16 | (C)16 | |  | 3 | D |  | 0.16X16 | =2.56 | (2)16 | |  |  |  |  | 0.56X16 | =8.96 | (8)16 |   = (3DC.DC28)16 | 14) Hexadecimal to Decimal  (ABC.CD)16=(?)10  = Ax162+ Bx161+ Cx160. Cx16-1+ Dx16-2  =(2748.80)10 |
| 15) Octal to Hexadecimal  (21056.375)8 = (?)16 = (222E.7E8)16   |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 2 | 1 | 0 | 5 | 6 | . | 3 | | 7 | | 5 | | 010 | 001 | 000 | 101 | 110 |  | 011 | | 111 | | 101 | | Append 0 | | | | | | | | | | | | 0010 | 0010 | 0010 | 1110 | . | 0111 | | 1110 | | 1000 | | | 2 | 2 | 2 | E | . | 7 | | E | | 8 | | | 16) Hexadecimal to Octal  (222E.7E8)16= (?)8  = (21056.375)8   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | 2 | 2 | 2 | E | . | 7 | E | 8 | | 0010 | 0010 | 0010 | 1110 |  | 0111 | 1110 | 10000 | |  |  |  |  |  |  |  |  | | 010 | 001 | 000 | 101 | 110 | . | 011 | 111 | | 2 | 1 | 0 | 5 | 6 | . | 3 | 7 | |
| 17) Convert  (284.65)10 = ( )8 = ( )16   |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 8 | 284 |  |  | 0.65x8 | = | 5.20 | = | 5 | | 8 | 35- | 4 | 0.20x8 | = | 1.60 | = | 1 | |  | 4 | 3 | 0.60x8 | = | 4.80 | = | 4 | |  | | | 0.80x8 | = | 6.40 | = | 6 | | 0.40x8 | = | 3.20 | = | 3 |   (284)10=(434)8 (0.65)10=(0.51463)8  (284.65)10= (434.51463)8  (284.65)10 = (11C.A666)16   |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 16 | 284 |  |  | 0.65x16 | = | 10.40 | = | A | | 16 | 17 | 12 | 0.40x16 | = | 6.40 | = | 6 | |  | 1 | 1 | 0.40x16 | = | 6.40 | = | 6 | |  | | | | 0.40x16 | = | 6.40 | = | 6 | | Convert the following  1) (F9AC.5D8B)16 =( ?)10  2) (457.245)8 = (?)10  3) (526.44)8 = (?)2 = (?)10  4) (48350)10 = (?)16 = (?)8 |

**1.5 Algorithms and Flowcharts**

**1.5.1 Algorithms**

An algorithm can be defined as a sequence of steps or instructions to solve a problem.

To make a computer do anything, you have to write a computer program. To write a computer program, you have to tell the computer, step by step, exactly what you want it to do. The computer then "executes" the program, following each step, to accomplish the end goal.

When you are telling the computer what to do, you also get to choose how it's going to do it. That's where **computer algorithms** come into picture. The algorithm is the basic technique used to get the job done. Let's follow an example to help get an understanding of the algorithm concept.

Let's say that you have a friend arriving at the airport, and your friend needs to get from the airport to your house. Here are four different algorithms that you might give your friend for getting to your home:

**The taxi algorithm**:

1. Go to the taxi stand.
2. Get in a taxi.
3. Give the driver my address.

**The call-me algorithm**:

1. When your plane arrives, call my cell phone.
2. Meet me outside baggage claim.

**The rent-a-car algorithm**:

1. Take the shuttle to the rental car place.
2. Rent a car.
3. Follow the directions to get to my house.

**The bus algorithm**:

1. Outside baggage claim, catch bus number 70.
2. Transfer to bus 14 on Main Street.
3. Get off on Elm street.
4. Walk two blocks north to my house.

All four of these algorithms accomplish exactly the same goal, but each algorithm does it in completely different way. Each algorithm also has a different cost and a different travel time. Taking a taxi, for example, is probably the fastest way, but also the most expensive. Taking the bus is definitely less expensive, but a whole lot slower. You choose the algorithm based on the circumstances.

**1.5.2 Flow chart**

A flowchart is a pictorial representation of an algorithm, workflow or a process, showing the steps as boxes of various kinds, and their order by connecting them with arrows. This diagrammatic representation illustrates a solution model to a given problem.

Or pictorial representation of solution to a given problem. Figure 1.4 shows the different components of a flowchart.

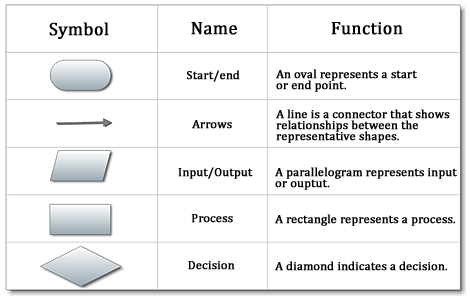


Fig 1.4 Components of flow chart

**Problem 2: find an area of circle (Area= 3.14\*r\*r) of radius r**

Spet1: Start

Step2: Read /input r

Step3: Area=3.14\*r\*r

Step4: Print/Output Area

Step5: Stop

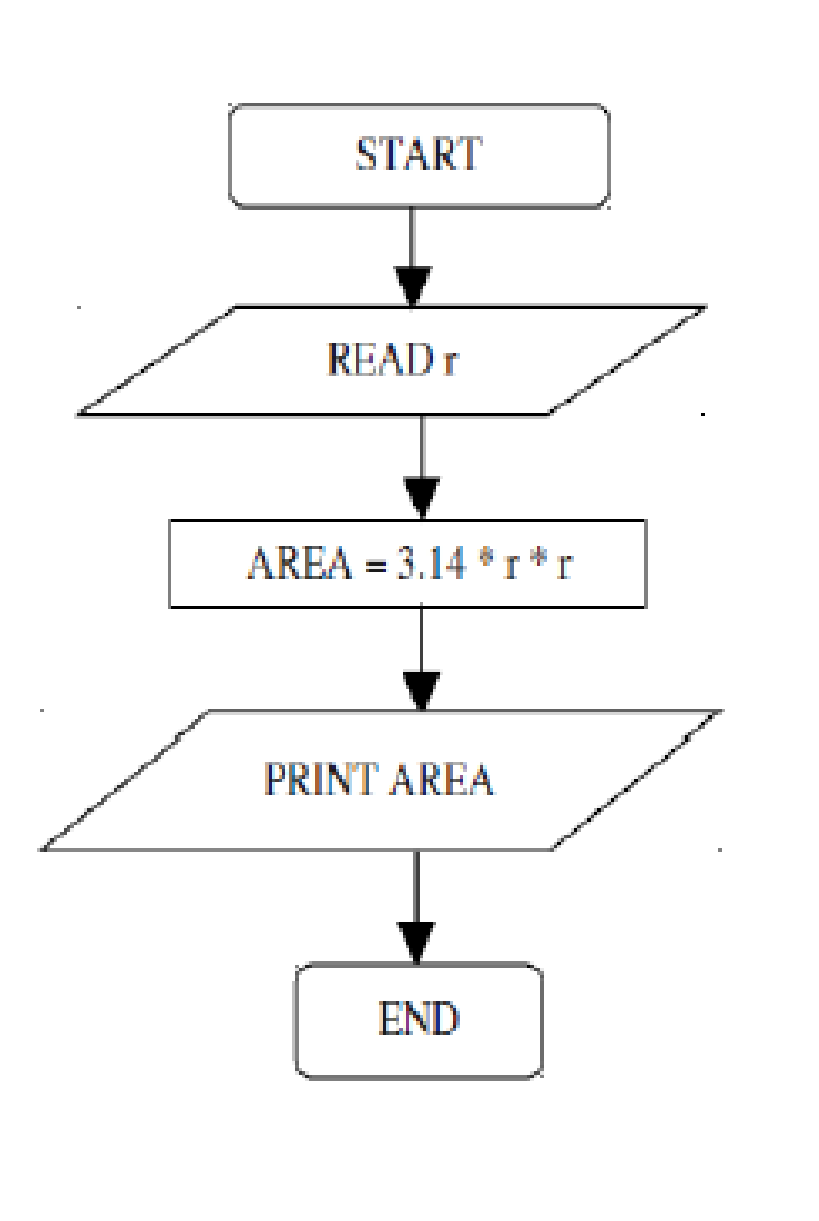


Fig 1.6 flow chart to find the area of a circle

***Problem1: write algorithm to find the greater number between two numbers***

Step1: Start

Step2: Read/input A and B

Step3: If A greater than B then

Step4: Print A is big

Step5: else print B is big

Step6: End

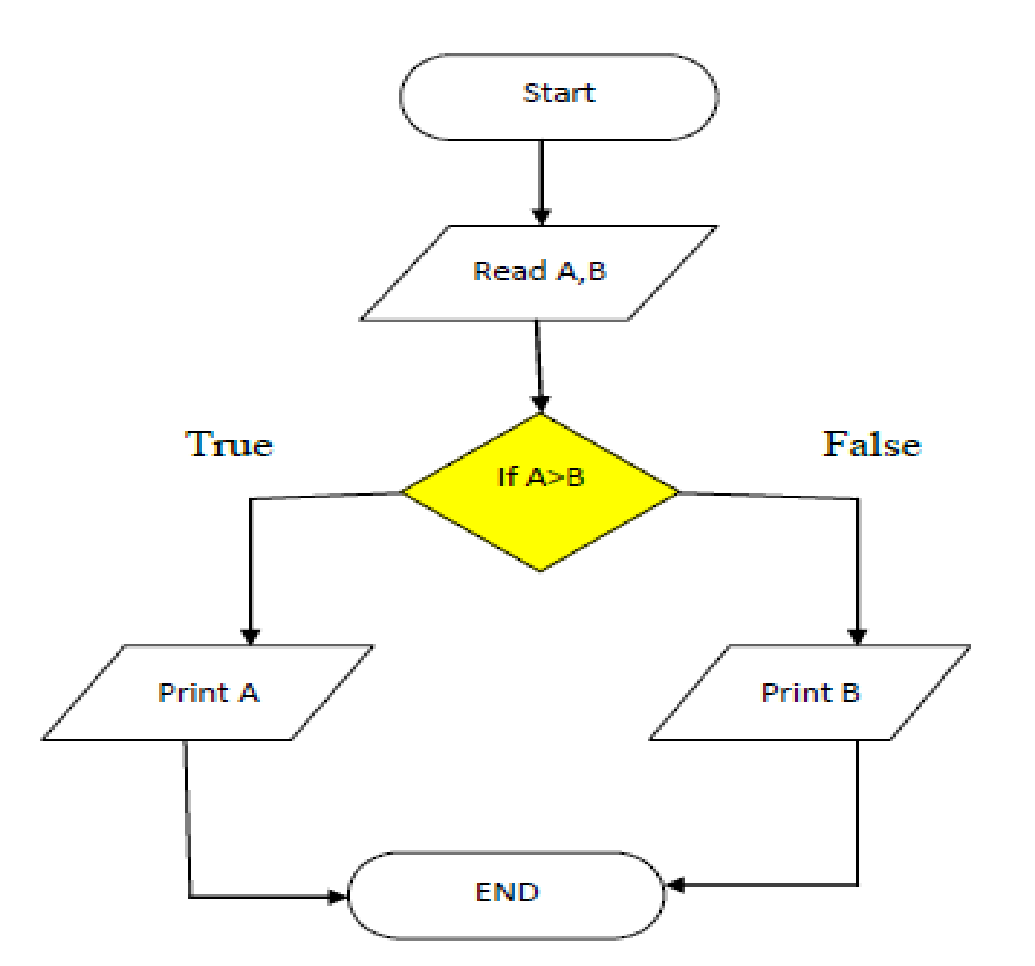


Fig 1.7 Flow chart to find the largest of two numbers

Flowchart to find largest of 3 no’s

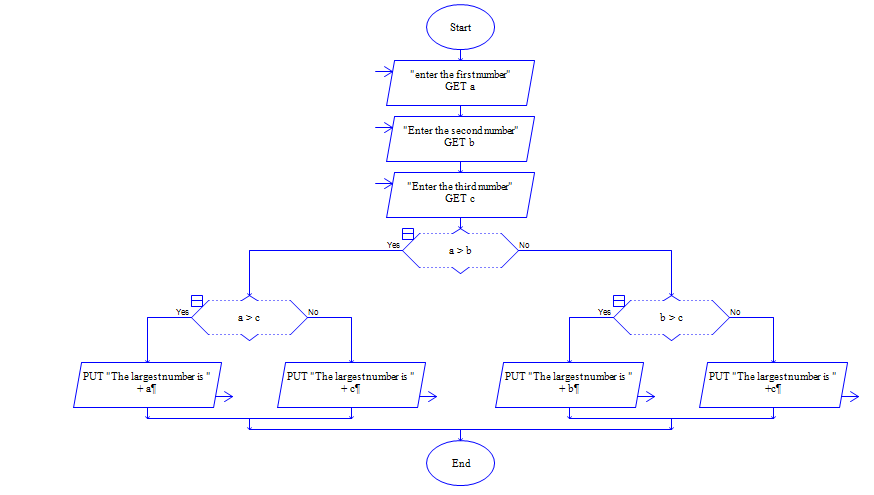
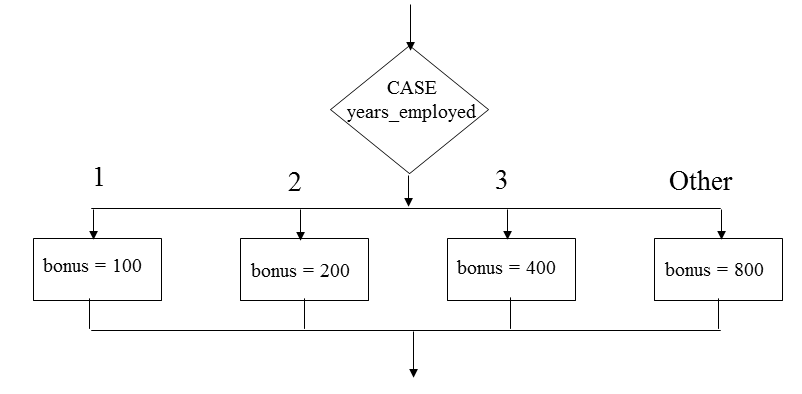
****

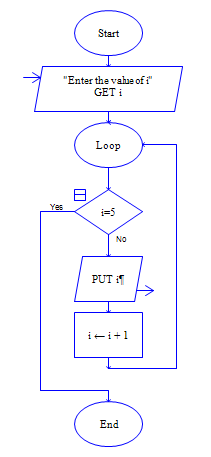
Fig 1.8 Flowchart to find largest of 3 no’s developed using RAPTOR tool

Flow chart for SWITCH-CASE:



Above figure demonstrates the flowchart for switch-case for an example where an employee is awarded bonus based on the number of years of employment in an organization. An employee with one year of experience gets a bonus of 100, two years gets 200 and so on. A default case of “Others” is considered where an employee gets a bonus of 800.

Flowchart in case of looping:



Above figure demonstrates a flowchart in case of looping constructs such as “while” and “for” loops. A variable such as “i” is initialized with a value. This variable is tested for a condition, the statements in the loop (i=i+1) keeps executing until the condition i=5 is satisfied. It should also be noted that the instructions in the loop gets executed until the condition is satisfied (in this case PUT i, to print the value of gets printed until the condition is satisfied).

**Chapter 2: C Programming Basics**

**2.1 Structure of a C Program**

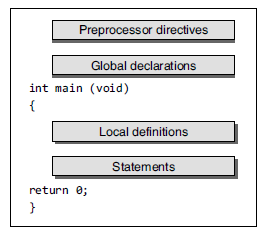
****

Figure 2.1: Structure of a C Program

Declaration is the program statement that serves to communicate to the language translator information about the name and type of the data objects needed during program execution.

The preprocessor directives tell the preprocessor to look for special code libraries, make substitutions in the code, and in other ways prepare the code for translation into machine language.

Global declarations involves declaration of the variables that are visible to all the parts of the program.

All functions including main() can be divided into two sections—local definition and statements. Local definitions would be at the beginning of the functions which is followed

by statement section. It describes the data that will be used in the function. Data objects in local definitions as opposed to global declarations are visible only to the function that contains them. Statement section consists of the instructions that cause the computer to do something.

The difference between a declaration and a definition is important. A declaration announces the properties of a data object or a function. The main reason for declaring data objects and functions is type checking. If a variable or function is declared and later reference is made to it with data objects that do not match the types in the declaration, the compiler will complain.

A definition, on the other hand, actually sets aside storage space (in the case of a data object) or indicates the sequence of statements to be carried out (in the case of a function).

**2.2. Basic Execution of a C Program**

The steps involved in execution of a basic program are as listed as depicted in figure 2.3

We shall consider execution of simple program written as shown and explain with reference this program.

The program is written using a text editor such as notepad and the file is saved as simple.c

The “.c” extension is used to indicate that the file is a C program

File: simple.c

#include <stdio.h>

void main()

{

printf(“Hello C Language”);

}

Let’s try to understand the flow of above program

*Step 1: Create/Edit*

1. This is First Step is Creating and Editing Program.

2. Firstly, we have to write a C program using an editor.

3. Save the Program by using [.C] Extension.

4. File Saved with [.C] extension is called “Source Program “.

*Step 2: Compiling*

1. Compiling C Program: C Source code with [.C] Extension is given as input to compiler such as Turbo C and compiler converts it into Equivalent Machine Instruction.

2. Compiler Checks for errors. If source code is error-free then Code is converted into Object File [.obj ].

*Step 3: Checking Errors*

1. During Compilation Compiler will check for error, if compiler finds any error then it will report it.

2. User has to go back to the source program and edit the program to correct the errors.

3. After editing and saving the program, It is compiled again to check for any errors.

4. If program is error-free then program is linked with appropriate libraries.

*Step 4: Linking Libraries*

1. The program is linked with included header files.

2. The program is linked with other libraries.

3. The process of linking the source program with libraries and header files is achieved using a software called Linker.

*Step 5: Error Checking*

1. Once the program is compiled and linked, an executable “.exe” file is generated. The program is executed using this “.exe” file. The user has to thoroughly test the program for different input combinations to identify and nalyse if there are any runtime errors (such as divide by 0).

2. If there have been any run time errors then the programmer has to check the code and find a solution to overcome the run time error. The program has to be tested again until it is error free.

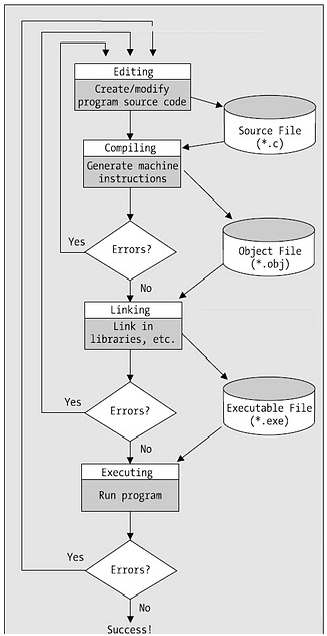


Figure 2.2 Basic execution flow of a C program

**2.3Variables**

A variable is a name given to a storage area that our programs can manipulate. Each variable in C has a specific type, which determines the size and layout of the variable’s memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable. The name of a variable can be composed of letters, digits, and the underscore character. It must begin with either a letter or an underscore. Upper and lowercase letters are distinct because C is case-sensitive.

**Syntax**

*Data\_type variable\_list;*

Here, type must be a valid C data type including char, int, float, double, bool, or any user-defined object; and variable\_list may consist of one or more identifier names separated by commas.

Example:

int I;

float weight;

char section;

here the data types in the above three declarations are int for type integer, float for the type real and char for character data type are declared using the keywords. The variable names are “i”, “weight” and “section”.

**Rules for naming a variable in C**

1. A variable name can have letters (both uppercase and lowercase letters), digits and underscore only.
2. The first letter of a variable should be either a letter or an underscore. However, it is discouraged to start variable name with an underscore. It is because variable name that starts with an underscore can conflict with system name and may cause error.
3. There is no rule on how long a variable can be. However, only the first 31 characters of a variable are checked by the compiler. So, the first 31 letters of two variables in a program should be different.

**2.4 Constants/Literals**

A constant is a value or an identifier whose value cannot be altered in a program. An identifier also can be defined as a constant.

**example**

*const double PI = 3.14*

**2.5 Operators**

**Arithmetic Operators**

An arithmetic operator performs mathematical operations such as addition, subtraction and multiplication on numerical values (constants and variables).

**Operator Meaning of Operator**

+ addition or unary plus

- subtraction or unary minus

\* multiplication

/ division

% remainder after division( modulo division)

Examples of using arithmetic operators:

Assume two integer variables a and b with values 5 and 14 respectively.

An expression a + b results in 5+14 = 19

An expression b/a results in a quotient 14/5 = 2 (result is 2 and not 2.8 because we are dividing two integers and the result is also an integer)

An expression b % a results in a remainder 14 % 5 = 4

An expression a\*b results in 5\*14 = 70

An expression a-b results in 5-14 = -9

Sample program:

void main()

{

int a, b, result;

printf(“Enter the values of a and b \n”);

scanf(“%d%d”,&a,&b);

result=a+b;

printf(“The sum of two numbers is %d”,c);

result=a+b;

printf(“The sum of two numbers is %d”,c);

result=a+b;

printf(“The sum of two numbers is %d”,c);

result=a+b;

printf(“The sum of two numbers is %d”,c);

result=a+b;

printf(“The sum of two numbers is %d”,c);

}

**Increment and decrement operators**

C programming has two operators increment ++ and decrement -- to change the value of an operand (constant or variable) by 1. Increment ++ increases the value by 1 whereas decrement -- decreases the value by 1. These two operators are unary operators, meaning they only operate on a single operand.

**Example**

#include <stdio.h>

int main()

{

int a = 10, b = 100;

float c = 10.5, d = 100.5;

printf("++a = %d \n", ++a);

printf("--b = %d \n", --b);

printf("++c = %f \n", ++c);

printf("--d = %f \n", --d);

return 0;

}

**Output**

++a = 11

--b = 99

++c = 11.500000

++d = 99.500000

++a refers to pre increment operator. The value of “a” is incremented by 1 before the statement is executed. Similarly C programming also has post increment operators such as a++ and a--.

The value of a gets incremented by 1 after the execution of the statement.

Example:

#include <stdio.h>

void main()

{

// program to demonstrate pre and post increment operators

int a=5;

printf("value of a before increment is %d \n",a++);

// increment within printf() does not get reflected in that printf() statement and output a=5

a++;

printf("Value of a after incrementing is %d \n",a);

}

/\* output

value of a before increment is 5

Value of a after incrementing is 7

\*/

**Assignment Operators**

An assignment operator is used for assigning a value to a variable. The assignment operator is =

**Operator Example Same as**

= a = b a = b

+= a += b a = a+b

-= a -= b a = a-b

\*= a \*= b a = a\*b

/= a /= b a = a/b

%= a %= b a = a%b

Example Program:

#include <stdio.h>

void main()

{

int a,b;// program to demonstrate pre and post increment operators

a=5;

b=3;

//sum

a+=b;

printf("Sum a+=b is %d \n",a);

a=5;

a\*=b;

printf("Sum a\*=b is %d \n",a);

a=5;

a/=b;

printf("Sum a/=b is %d \n",a);

a=5;

a%=b;

printf("Sum a mod b is %d \n",a);

a=5;

a-=b;

printf("Sum a-=b is %d \n",a);

}

/\* Output

Sum a+=b is 8

Sum a\*=b is 15

Sum a/=b is 1

Sum a mod b is 2

Sum a-=b is 2

\*/

**Relational Operators**

A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0. Relational operators are used in decision making and loops.

**Operator Meaning of Operator Example**

== Equal to 5 == 3 returns 0

> Greater than 5 > 3 returns 1

< Less than 5 < 3 returns 0

!= Not equal to 5 != 3 returns 1

>= Greater than or equal to 5 >= 3 returns 1

<= Less than or equal to 5 <= 3 return 0

**Logical Operators**

An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false. Logical operators are commonly used in decision making in C programming.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Meaning of Operator** | **Example** |
| && | Logial AND. True only if all operands are true | If c = 5 and d = 2 then, expression ((c = 5) && (d > 5)) equals to 0. |
| || | Logical OR. True only if either one operand is true | If c = 5 and d = 2 then, expression ((c == 5) || (d > 5)) equals to 1. |
| ! | Logical NOT. True only if the operand is 0 | If c = 5 then, expression! (c == 5) equals to 0. |

**Bitwise Operators**

During computation, mathematical operations like: addition, subtraction, addition and division are converted to bit-level which makes processing faster and saves power. Bitwise operators are used in C programming to perform bit-level operations.

**Operators Meaning of operators**

& Bitwise AND

| Bitwise OR

^ Bitwise exclusive OR

~ Bitwise complement

<< Shift left

>> Shift right

**The sizeof operator**

The *sizeof* is an unary operator which returns the size of data (constant, variables, array, structure etc).

**Example**

#include <stdio.h>

int main()

{

int a, e[10];

float b;

double c;

char d;

printf("Size of int=%lu bytes\n",sizeof(a));

printf("Size of float=%lu bytes\n",sizeof(b));

printf("Size of double=%lu bytes\n",sizeof(c));

printf("Size of char=%lu byte\n",sizeof(d));

printf("Size of integer type array having 10 elements = %lu bytes\n", sizeof(e));

return 0;

}

**Output**

Size of int = 4 bytes

Size of float = 4 bytes

Size of double = 8 bytes

Size of char = 1 byte

Size of integer type array having 10 elements = 40 bytes

**Ternary Operator (?:)**

A conditional operator is a ternary operator, that is, it works on 3 operands.

**Syntax**

*conditionalExpression ? expression1 : expression2*

**Explanation**

* The first expression conditionalExpression is evaluated first. This expression evaluates to 1 if it's true and evaluates to 0 if it's false.
* If conditionalExpression is true, expression1 is evaluated.
* If conditionalExpression is false, expression2 is evaluated.

**2.6 Input / Output statements**

**scanf() – Input statement**

* This function is usually used as an input statement.
* The format string must be a text enclosed in double quotes. It contains the information for interpreting the entire data for connecting it into internal representation in memory.
  + *Example: integer (%d), float (%f), character (%c) or string (%s)*
* The argument list contains a list of variables each proceeded by the address list and separated by comma.
* The number of argument is not fixed; however corresponding to each argument there should be a format specifier. Inside the format string the number of argument should tally with the number of format specifier.

**Syntax**

*scanf (“format string”, argument list);*

**printf () – Output statement**

This function is used as an output statement.

**Syntax**

*printf (“format string”, argument list);*

**Format specifiers in C**

|  |  |  |
| --- | --- | --- |
| **type** | **Qualifying Input** | **Type of argument** |
| c | Single character: Reads the next character. If a width different from 1 is specified, the function reads width characters and stores them in the successive locations of the array passed as argument. No null character is appended at the end. | char \* |
| d | Decimal integer: Number optionally preceded with a + or - sign | int \* |
| e, E, f, g, G | Floating point: Decimal number containing a decimal point, optionally proceeded by a + or - sign and optionally followed by the e or E character and a decimal number. Two examples of valid entries are -732.103 and 7.12e4 | float \* |
| o | Octal Integer: | int \* |
| s | String of characters. This will read subsequent characters until a whitespace is found (whitespace characters are considered to be blank, newline and tab). | char \* |
| u | Unsigned decimal integer. | unsigned int \* |
| x, X | Hexadecimal Integer | int \* |

**Escape sequences in C**

**Escape sequence Meaning**

\n New line

\t Tab

\b Back space

\a Bell

\o Null character

\? To print question mark

\\ To print slash

\’ To print single quote

\” To print double quote

**2.7 Selection statements**

C has two types of [selection statements](https://en.wikipedia.org/wiki/Selection_statement)

* + if statement
  + switch statement.

**if statement**

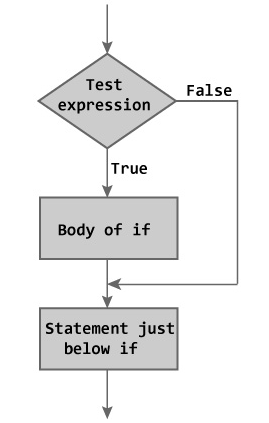
**Syntax**:

**if** (<expression)>

<statement 1>

**Explanation**

The if statement evaluates the test expression inside the parenthesis. If the test expression is evaluated to true (nonzero), statements inside the body of if is executed. If the test expression is evaluated to false (0), a statement inside the body of if is skipped from execution. The flow of work for if statement is as follows:

****

**if-else statement**

It is the basic form where the if statement evaluate a test condition and direct program execution depending on the result of that evaluation.

**Syntax**:

**if** (<expression)>

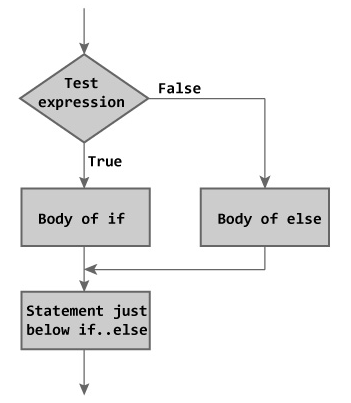
<statement 1>

**else**

<statement 2>

**Explanation**

In if statement, if the <expression> in parentheses is nonzero (true), control passes to <statement1>. If the else clause is present and the <expression> is zero (false), control will pass to <statement2>. The else <statement2> part is optional and, if absent, a false <expression> will simply result in skipping over the <statement1>. The flow of work for if statement is as follows:



**Nested if...else statement**

**Syntax**

**if** (testExpression1)

{

// statements to be executed if testExpression1 is true

}

**else if** (testExpression2)

{

// statements to be executed if testExpression1 is false and testExpression2 is true

}

**else if** (testExpression 3)

{

// statements to be executed if testExpression1 and testExpression2 is false and testExpression3 is true

}

.

.

**else**

{

// statements to be executed if all test expressions are false

}

The ***if...else*** statement executes two different codes depending upon whether the test expression is true or false. Sometimes, a choice has to be made from more than 2 possibilities. The *nested if...else* statement allows you to check for ***multiple test expressions*** and execute different codes for more than two conditions.

**switch statement**

This is another form of the multi way decision. It is well structured, but can only be used in certain cases where;

* Only one variable is tested, all branches must depend on the value of that variable and the variable must be an integral type. (int, long, short or char).
* Each possible value of the variable can control a single branch. A final, catch all, default branch may optionally be used to trap all unspecified cases.

**Syntax**:

switch (n)

​{

case constant1:

// code to be executed if n is equal to constant1;

break;

case constant2:

// code to be executed if n is equal to constant2;

break;

.

.

.

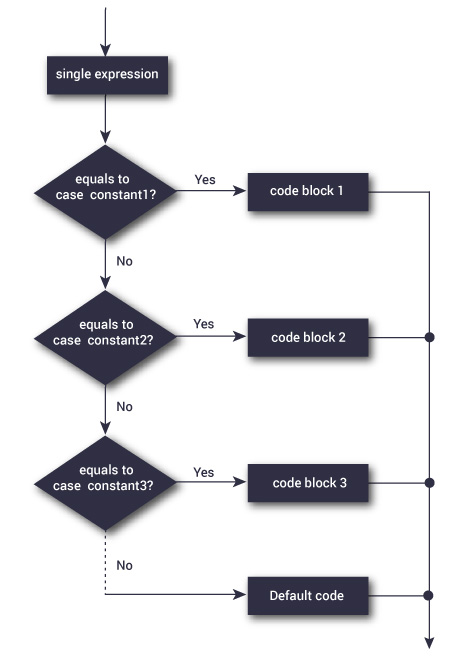
default:

// code to be executed if n doesn't match any constant

}

**Explanation**

The switch statement causes control to be transferred to one of several statements depending on the value of an expression, which must have integral type. The substatement controlled by a switch is typically compound. Any statement within the substatement may be labeled with one or more case labels, which consist of the keyword case followed by a constant expression and then a colon (:). The work flow of switch case statement is as follows:



**2.7 Loops**

Looping is a way by which we can execute any some set of statements more than one times continuously .In c there are mainly three types of loops are use :

* while Loop
* do while Loop
* for Loop

**while Loop**

**Syntax**

Initialization

while ( <*expression>* )

{

<*Statement1>*

<*Statement2>*

<*Statement3>*

}

The most basic *loop* in C is the while loop. A while loop has one control expression, and executes as long as that expression is true. The while loops works in the following way:



Loops generally consist of two parts: one or more *control expressions* which control the execution of the loop, and the *body*, which is the statement or set of statements which is executed over and over.

**do while Loop**

**Syntax**

Initialization

do

{

<*Statement1>*

<*Statement2>*

<*Statement3>*

} while ( <*expression>* )

**Explanation**

This is very similar to the while loop except that the test occurs at the end of the loop body. This guarantees that the loop is executed at least once before continuing. Such a setup is frequently used where data is to be read. The test then verifies the data, and loops back to read again if it was unacceptable.



**for loop**

**Syntax**

for ( <initialization> ; <condition> ; <increment / decrement> )

<statement>

**Explanation**

**initialisation** :- this is usually an assignment to set a loop counter variable for example.

**condition** :- determines when loop will terminate.

**increment** :- defines how the loop control variable will change each time the loop is executed.

**statement body** :- can be a single statement, no statement or a block of statements.

The for statement executes as follows :-



***Conditional Operator :- ?:***

***Syntax***

condition ? expr\_1 : expr\_2 ;

This is a special shorthand operator in C and replaces the following segment

if ( condition )

expr\_1 ;

else

expr\_2 ;

**Explanation**

The ?: operator is a ternary operator in that it requires three arguments. One of the advantages of the ?: operator is that it reduces simple conditions to one simple line of code which can be thrown unobtrusively into a larger section of code.

Example:

#include <stdio.h>

void main()

{

int a,b;// program to demonstrate pre and post increment operators

a=5;

b=3;

printf("%d",a>b?a:b);

}

The output of the above program is 5. The conditional operator ?: is used within the printf(). If a>b is true then the value of “a” is printed, else the value of “b” is printed. Since a>b is true the output value 5 gets printed on executing this program.

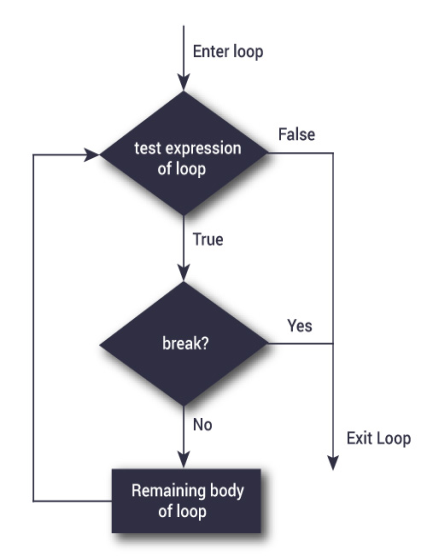
**break statement**

It is sometimes desirable to skip some statements inside the loop or terminate the loop immediately without checking the test expression. The break statement terminates the loop (for, while and do..while) immediately when it is encountered. The break statement can be used with decision making statement such as if..else

**Syntax**

break;

**workflow of break statement**

****

**How break statement works?**

|  |  |
| --- | --- |
|  |  |

**continue statement**

The continue statement skips some statements inside the loop. The continue statement is used with decision making statement such as if...else.

**syntax**

continue;

**How continue statement works?**

|  |  |
| --- | --- |
|  |  |