DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COURSE MATERIAL

Faculty Details

Name of the Faculty	:	V.BALU
Designation	:	Assistant Professor
Department	:	CSE

Course Details

Name of the Course	:	BE
Branch	:	ECE / EEE / EIE / MECT
Year	:	I
Subject Code	:	CESCS18T40
Title of the Subject	:	PROGRAMMING FOR PROBLEM SOLVING
Batch	:	2020-2021

Course Code: CESCS18T40	**PROGRAMMING FOR PROBLEM SOLVING**	**L**	**T**	**P**	**Credit**	CIA Marks **40**
Course Category: ESC		2	1	0	3	SEE Marks **60**

COURSE OBJECTIVES:

- Be exposed to the syntax of C.
- Be familiar with programming in C.
- Learn to use arrays, strings, functions, pointers, structures and unions in C.

COURSE OUTCOMES:

- Design C Programs for problems.
- Write and execute C programs for simple applications
- Design problem solutions using Object Oriented techniques

After the successful completion of the course students will be able to

COS NO.	**Course Outcomes**	**Bloom's level**
CO1	Develop algorithms for solving simple mathematical and engineering problems and examine the suitability of appropriate repetition and/or selection structures for given problems	Apply
CO2	Solve matrix problems, merging, searching, sorting and string manipulation problems using iteration, modularization or recursion as applicable.	Apply
CO3	Organize files to perform text operations like editing, pattern searching using structures	Apply
CO4	Implement the algorithms for matrix problems, merging, searching, sorting, and string manipulation and file problems and debug and test using any procedural programming language	Apply

Mapping of Course Outcome to Program Outcomes:

Course Outcomes	Program Outcomes											
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	S	S	S	S	S	M	L	L	M	M	-	M
CO2	S	S	S	S	S	M	L	L	M	M	-	M
CO3	S	S	M	M	S	M	L	L	M	M	-	M
CO4	S	S	S	S	S	M	L	L	S	S	-	S

SYLLABUS:

MODULE I

Introduction to components of computer system-Generation of programming languages-Types of Computers-Organization of Computers-Types of memory, Number systems-Idea of Algorithm-Pseudo code- Flow Chart with examples.

MODULE II

Introduction to C-Character set, Constants, Variables, Data Types-Operators – Arithmetic expressions and precedence-Decision Making statement - Looping statements.

MODULE III

Arrays and its types-Functions –Parameter passing in functions-call by value- call by reference – Passing array to functions-Recursive function.

MODULE IV

Structures and array of structures –Union, Basic searching –Linear and Binary, Basic sorting, String operations.

MODULE V

Introduction to Pointer, Pointer arithmetic-notion of linked list (no implementation) - File handling.

LEARNING RESOURCES:

Text Books:

1. Byron Gottfried, Schaum‟s Outline of Programming with C, McGraw-Hill.4th Edition july 2018.
2. Balagurusamy. E, “Programming in ANSI C”, Tata McGraw Hill, Third edition, 2008
3. Fundamentals of Computing and Programming- V.Ramesh Babu, R.Samyuktha, M.Muniratham by VRB Publishers Sep 2018 edition.

References:

1. Let Us 'C' - Yashawant Kanetkar, (Unit 2 to 5), BPB publications, 16th Edition, 2017.
2. Ashok N Kamthane, “Computer Programming”, Pearson education, Second Impression, 2011.
3. Venugopal. K and Kavichithra.C, “Computer Programming”, New Age International Publishers, 2nd Edition,Jan 2008.
4. Kernighan B.W and Ritchie,D.M , The C programming language: second edition, Pearson education, 2015.

Online resources:

1. www.nptl.co.in
2. www.electrical4u.com

ASSESSMENT PATTERN:

Bloom’s Category	**Continuous Assessment Tests**		**Assignment (10)**	**Terminal Examination (60)**
	(15)	**(15)**		
Remember	7	0	10	20
Understand	8	-		15
Apply	0	8		10
Analysis	0	4		10
Evaluate	0	3		5
Create	0	0		0
Total	**15**	**15**		**60**
	A	**B**	**C**	**D**

A+B+C+D = 100

COURSE PLAN

Select a course (a theory subject): Programming for Problem Solving

Select a unit: Unit I

Select a Topic: Introduction to components of computer system, Number System, Algorithm and Flow Chart.

1. **Objectives**

   To Study the Basics of Computers, its organization and its Generations

2. **Outcomes**

   Students will acquire knowledge on Basics of computers, Evolution of Computers, Organization of computers, Generation of Computers

3. **Pre-requisites**

   Nil

4. **Terminology used other than normal known scientific / engg terms and their fundamental explanations / relations**

Plan for the lecture delivery

1. **How to plan for delivery – black board / ppt / animated ppt / (decide which is good for this topic)**

   Power Point Presentation

2. **How to explain the definition and terms with in it**

   Describe the functions of computers and history of Computers.

3. **How to start the need for any derivation / procedure / experiment / case study**

   Not Applicable

4. **Physical meaning of math equations / calculations**

   Not Applicable

5. **Units and their physical meaning to make them understand practical reality and comparison between different units**

   Understand the concept of various parts of the computer is useful for when writing program.

6. **What is the final conclusion?**

   Understand how computer works and how number system used in computer

7. **How to put it in a nut shell**

   Types of computers, Generations of Computers, Numbers System, Algorithm, Flow chart

8. **Important points for understanding / memorizing / make it long lasting**

   Computer is a Calculating Device, First Generation, second Generation, third Generation, fourth Generation and Fifth Generation, Number System Decimal , Binary, Octal, Hexadecimal. Algorithm is Step by Step solution to the Problem, Flow Chart is a Pictorial representation of algorithm.

9. **Questions and cross questions in that topic to make them think beyond the topic.**

A 1. Discuss in detail about
a) Characteristics of computer b) Evolution of computers

B 1.Define Parts of computer?
2.what is storage unit?

C 1. Explain in detail about Organization of Computer with Neat diagram.
2.Explain various generations of computers.

D 1. Classify and Explain the types of Computers based on configuration.
E 1. What are the types of computers?

F 1.Explain the types of Computers based on mode of   use.

10. **Final conclusions**

    Students will acquire knowledge on Basics of computers, Evolution of Computers, Organization of computers, Generation of Computers .

Introduction to digital Computers:

1Parts of Computer

Performance

Features that affect the performance of the computer include:

Microprocessor Operating System
RAM
Diskdrives
display input/output ports

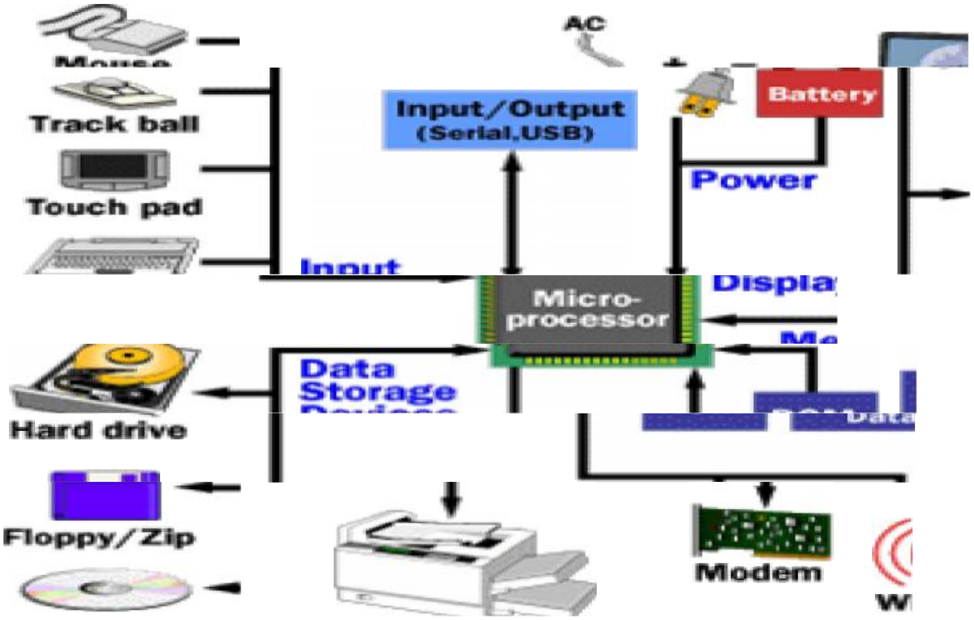


Fig: Schematic diagram showing the various parts of a computer.

The microprocessor:

- Has a set of internal instructions stored in memory, and can access memory for its own use while working.
- Can receive instructions or data from you through a keyboard in combination with another device
- (Mouse, touchpad, trackball, and joystick).
- Can receive and store data through several data storage devices (hard drive, floppy drive, Zip drive, CD/DVD drive).
- Can display data to you on computer monitors (cathode ray monitors, LCD displays).
- Can send data to printers, modems, networks and wireless networks through various input/output ports. Is powered by AC power and/or batteries.

Characteristics of Computers

The various characteristics of computers are as follows

1. **Speed**: Speed is the most important characteristics of computer .Computer having more speed to perform jobs instantaneously.

2. **Accuracy:** The computers are perfect, accurate and precise. Accuracy signifies the reliability of the hardware components of computers.

3. **Automatic:** A computer works automatically, once programs are stored and data are given to it, constant supervision is not required.

4. **Endurance:** A computer works continuously and will not get tired and will not suffer from lack of concentration.

5. **Versatility**: A computer can be put to work in various fields.

6. **Reduction of cost**: Though initial investment may be high, computer substantially reduces the cost of transaction.

Evolution of Computers

The Early development:

ABACUS:

ABACUS uses movable beads stung on wires above and below a cross bar and its operation are based on the idea of the place value notation. The beads of the counter represent digits. The value of the digit in each position is determined by adding the values of the beads pressed against the cross piece.

Pascal calculating machine:

It was the first real desktop calculating device that could add and subtract. It consists of a set of toothed wheels or gears with each wheel or gear having digits 0 to 9 engraved on it. Arithmetic operation could be performed by tunings these wheels.

Punched card Machine:

The presence and absence of the holes in the card represent the digits.

Charles Babbage’s Engine:

The machine took input from the punched card. The Analytical engine had a memory which will perform arithmetic operations.

Computer Generations

Generation in computer terminology is a change in technology a computer is/was being used. Initially, the generation term was used to distinguish between varying hardware technologies. But nowadays, generation includes both hardware and software, which together make up an entire computer system.

There are totally five computer generations known till date. Each generation has been discussed in detail along with their time period, characteristics. We've used approximate dates against each generations which are normally accepted.

Following are the main five *generations* of computers

S.N.	Generation & Description
1	**First Generation** The period of first generation : 1946-1959. Vaccum tube based.
2	**Second Generation** The period of second generation : 1959-1965. Transistor based.
3	**Third Generation** The period of third generation : 1965-1971. Integrated Circuit based.
4	**Fourth Generation** The period of fourth generation : 1971-1980. VLSI microprocessor based.
5	**Fifth Generation** The period of fifth generation : 1980-onwards.ULSI microprocessor based

First Generation

i. The period of first generation was 1946-1959.

ii. First generation of computer started with using vacuum tubes as the basic components for memory and circuitry for CPU (Central Processing Unit). These tubes like electric bulbs produced a lot of heat and were prone to frequent fusing of the installations, therefore, were very expensive and could be afforded only by very large organisations.

iii. In this generation mainly batch processing operating system were used. In this generation Punched cards, Paper tape, Magnetic tape Input & Output device were used.

iv. There were Machine code and electric wired board languages used.

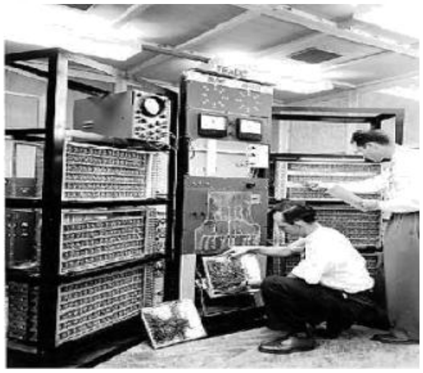


Fig: First Generation

The main features of First Generation are:

- Vacuum tube technology
- Unreliable
- Supported Machine language only
- Very costly
- Generate lot of heat
- Slow Input/Output device
- Huge size
- Need of A.C.
- Non portable
- Consumed lot of electricity

Some computer of this generation were:

- ENIAC
- EDVAC
- UNIVAC
- IBM-701
- IBM-650

Second Generation

i. The period of second generation was 1959-1965.

ii. This generation using the transistor were cheaper, consumed less power, more compact in size, more reliable and faster than the first generation machines made of vaccum tubes. In this generation, magnetic cores were

iii. used as primary memory and magnetic tape and magnetic disks as secondary storage devices.

iv. In this generation assembly language and high level programming language like FORTRAN, COBOL were used.

v. There were Batch processing and Multiprogramming Operating system used.

Fig: Second Generation

The main features of Second Generation are:

- Use of transistors
- Reliable as compared to First generation computers
- Smaller size as compared to First generation computers
- Generate less heat as compared to First generation computers
- Consumed less electricity as compared to First generation computers
- Faster than first generation computers
- Still very costly
- A.C. needed
- Support machine and assembly languages

Some computer of this generation were:

- IBM 1620
- IBM 7094
- CDC 1604
- CDC 3600
- UNIVAC 1108

Third Generation

i. The period of third generation was 1965-1971.

ii. The third generation of computer is marked by the use of Integrated Circuits (IC's) in place of transistors. A

iii. Single I.C has many transistors, resistors and capacitors along with the associated circuitry. The I.C was invented by Jack Kilby . This development made computers smaller in size, reliable and efficient.

iv. In this generation Remote processing, Time-sharing, Real-time, Multi-programming Operating System were used.High level language (FORTRAN-II TO IV, COBOL, PASCAL PL/1, BASIC, ALGOL-68 etc.) were used during this generation.

Fig: Third Generation

The main features of Third Generation are:

- IC used
- More reliable
- Smaller size
- Generate less heat
- Faster
- Lesser maintenance

- Still costly
- A.C needed
- Consumed lesser electricity
- Support high level language

Some computer of this generation were:

- IBM-360 series
- Honeywell-6000 series
- PDP(Personal Data Processor)
- IBM-370/168
- TDC-316

Fourth Generation

i. The period of Fourth Generation was 1971-1980.

ii. The fourth generation of computers is marked by the use of Very Large Scale Integrated (VLSI) circuits. VLSI circuits having about 5000 transistors and other circuit elements and their associated circuits on a single chip made it possible to have microcomputers of fourth generation. Fourth Generation computers became more powerful, compact, reliable, and affordable. As a result, it gave rise to personal computer (PC) revolution.

iii. In this generation Time sharing, Real time, Networks, Distributed Operating System were used.

iv. All the Higher level languages like C and C++, DBASE etc. were used in this generation.

Fig: Fourth Generation

The main features of Fourth Generation are:

- VLSI technology used

- Very cheap
- Portable and reliable
- Use of PC's
- Very small size
- Pipeline processing
- No A.C. needed
- Concept of internet was introduced
- Great developments in the fields of networks
- Computers became easily available

Some computer of this generation were:

- DEC 10
- STAR 1000
- CRAY-1(Super Computer)

Fifth Generation

i. The period of Fifth Generation is 1980-till date.

ii. In the fifth generation, the VLSI technology became ULSI (Ultra Large Scale Integration) technology, resulting in the production of microprocessor chips having ten million electronic components.

iii. This generation is based on parallel processing hardware and AI (Artificial Intelligence) software.

iv. AI is an emerging branch in computer science, which interprets means and method of making computers think like human beings.

v. All the Higher level languages like C and C++, Java, .Net etc. are used in this generation.

AI includes:

- Robotics
- Neural networks
- Game Playing

Fig: Fifth Generation

The main features of Fifth Generation are:

- ULSI technology
- Development of true artificial intelligence
- Development of Natural language processing
- Advancement in Parallel Processing
- Advancement in Superconductor technology
- More user friendly interfaces with multimedia features
- Availability of very powerful and compact computers at cheaper rates Some computer types of

this generation are:

- Desktop
- Laptop
- NoteBook
- UltraBook
- ChromeBook

1.5 Basic Computer Organization:

Introduction to digital computers

The architecture of the computers has not changed but the technology used to accomplish those operations may vary from one computer to another computer. The basic computer organization remains the same for all computer systems. In 1950's, it took a room full of vacuum tubes and equipment to perform the tasks that are now replaced with a single chip, not bigger than a child's thumb nail. Advances in semiconductor technology have reduced the Size of the chip.

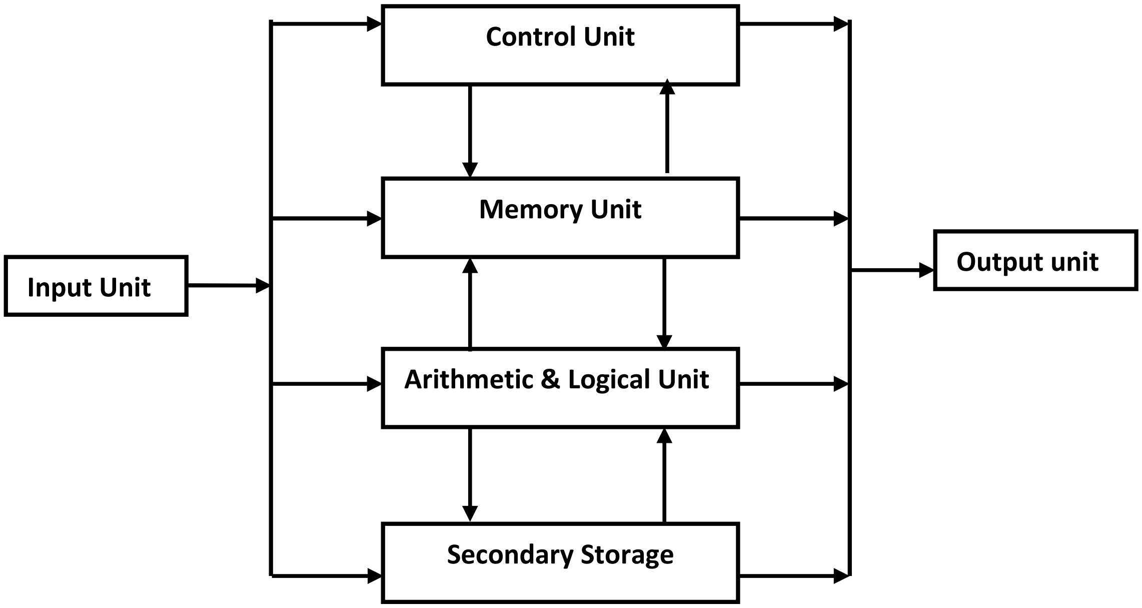


Fig: Architecture of computer

The block diagram of the digital computer system has the following three units.

1. Input unit
2. Central processing unit
3. Output unit

1. **Input unit**

Computers need data and instructions in order to solve any problem. We need to put the data and instructions into the computers. The input unit consists of one or more input devices. There are number of devices that perform the function of input devices. The keyboard of the computer is one of the most commonly used and standardized input device. Regardless of the type of input device used in a computer system, all input devices perform the following functions:

1. Accept data and instruction from the outside world.
2. Convert it to a form that the computer can understand.
3. Supply the converted data to the computer system for further processing.

The input unit is used to send information or instructions or commands to the computer. The input received from the input unit is immediately stored in main memory and the processed. Input may be processed. Input may be pressing a key in a keyboard or moving of mouse or movement of a joystick.

Following are few of the important input devices which are used in Computer Systems

- Keyboard
- Mouse
- Joy Stick
- Light pen
- Track Ball
- Scanner
- Graphic Tablet
- Microphone
- Magnetic Ink Card Reader (MICR)
- Optical Character Reader (OCR)
- Bar Code Reader
- Optical Mark Reader

Keyboard

Most common and very popular input device is keyboard. The keyboard helps in inputting the data to the computer. The layout of the keyboard is like that of traditional typewriter, although there are some additional keys provided for performing some additional functions.

Keyboards are of two sizes 84 keys or 101/102 keys, but now 104 keys or 108 keys keyboard is also available for Windows and Internet.

Fig: Keyboard

The keys are following

Sr. No.	Keys	Description
1	Typing Keys	These keys include the letter keys (A-Z) and dig its keys (0-9) which are generally give same layout as that of typewriters.
2	Numeric Keypad	It is used to enter numeric data or cursor movement. Generally, it consists of a set of 17 keys that are laid out in the same configuration used by most adding machine and calculators.
3	Function Keys	The twelve functions keys are present on the keyboard. These are arranged in a row along the top of the keyboard. Each function key has unique meaning and is used for some specific purpose.
4	Control keys	These keys provides cursor and screen control. It includes four directional arrow key. Control keys also include Home, End,Insert, Delete, Page Up, Page Down, Control(Ctrl), Alternate(Alt), Escape(Esc).
5	Special Purpose Keys	Keyboard also contains some special purpose keys such as Enter, Shift, Caps Lock, Num Lock, Space bar, Tab, and Print Screen.

Mouse

Mouse is most popular Pointing device. It is a very famous cursor-control device. It is a small palm size box with a round ball at its base which senses the movement of mouse and sends corresponding signals to CPU on pressing the buttons. Generally it has two buttons called left and right button and scroll bar is present at the mid. Mouse can be used to control the position of cursor on screen, but it cannot be used to enter text into the computer.

Fig: Mouse

Joystick

Joystick is also a pointing device which is used to move cursor position on a monitor screen. It is a stick having a spherical ball at its both lower and upper ends. The lower spherical ball moves in a socket. The Joystick can be moved in all four directions.

The function of joystick is similar to that of a mouse. It is mainly used in Computer Aided Designing (CAD) and playing computer games.

Fig: Joystick

Light Pen

Light pen is a pointing device which is similar to a pen. It is used to select a displayed menu item or draw pictures on the monitor screen. It consists of a photocell and an optical system placed in a small tube.

When light pen's tip is moved over the monitor screen and pen button is pressed, its photocell sensing element detects the screen location and sends the corresponding signal to the CPU.

Fig: Light Pen

Track Ball

Track ball is an input device that is mostly used in notebook or laptop computer, instead of a mouse. This is a ball which is half inserted and by moving fingers on ball, pointer can be moved.

Since the whole device is not moved, a track ball requires less space than a mouse. A track ball comes in various shapes like a ball, a button and a square.

Fig: Track Ball

Scanner

Scanner is an input device which works more like a photocopy machine. It is used when some information is available on a paper and it is to be transferred to the hard disc of the computer for further manipulation.

Scanner captures images from the source which are then converted into the digital form that can be stored on the disc. These images can be edited before they are printed.

Fig: Scanner

Digitizer

Digitizer is an input device which converts analog information into a digital form. Digitizer can convert a signal from the television camera into a series of numbers that could be stored in a computer. They can be used by the computer to create a picture of whatever the camera had been pointed at. Digitizer is also known as Tablet or Graphics Tablet because it converts graphics and pictorial data into binary inputs. A graphic tablet as digitizer is used for doing fine works of drawing and images manipulation applications.

Fig: Digitizer

Microphone

Microphone is an input device to input sound that is then stored in dig ital form. The microphone is used for various applications like adding sound to a multimedia presentation or for mixing music.

Fig: MicroPhone

Magnetic Ink Card Reader (MICR)

MICR input device is generally used in banks because of a large number of cheques to be processed every day. The bank's code number and cheque number are printed on the cheques with a special type of ink that contains particles of magnetic material that are machine readable.

This reading process is called Magnetic Ink Character Recognition(MICR). The main advantages of MICR are that it is fast and less error prone.

Fig: Magnetic Ink Card Reader

Optical Character Reader (OCR)

OCR is an input device used to read a printed text. OCR scans text optically character by character, converts them into a machine readable code and stores the text on the system memory.

Fig: Optical Character Reader

Bar Code Readers

Bar Code Reader is a device used for reading bar coded data (data in form of light and dark lines). Bar coded data is generally used in labelling goods, numbering the books etc. It may be a hand held scanner or may be embedded in a stationary scanner.

Bar Code Reader scans a bar code image, converts it into an alphanumeric value which is then fed to the computer to which bar code reader is connected.

Fig: Bar Code Reader

Optical Mark Reader (OMR)

OMR is a special type of optical scanner used to recognize the type of mark made by pen or pencil. It is used where one out of a few alternatives is to be selected and marked. It is specially used for checking the answer sheets of examinations having multiple choice questions.

Fig: Optical Mark Reader

2. **Central processing unit**
   It is the heart of the computer system. All operations are carried out in this unit only. In most modern computers, a single IC does the following job
   i) It performs all calculations
   ii) It makes all decisions
   iii) It controls all units of the computer

The CPU is sub-divided into the following sub-system:

- Control unit
- Arithmetic and logical unit (ALU)
- Memory unit
- Secondary storage devices.

(i) Control unit

The control unit instructs the computer how to carry out program instructions. It directs the flow of data between memory and arithmetic logical unit. It controls and coordinates the entire computer system.

The control unit controls all other parts of the computer. The input unit does not know when to receive data and where to put the data in the storage unit after receiving it. The control unit gives the necessary instructions to the input unit, the control unit instructs the input unit where to store the data after receiving it from user. In the same way, it controls the flow of data and instructions from the storage unit to ALU. It also controls the flow of the results from the ALU to the storage unit. The control unit also controls what should be sent to the output unit.

(ii) Arithmetic and logical unit

Arithmetic and logical unit performs all the arithmetic and logical operations, arithmetic operations like addition, subtraction, multiplication and logical operations such as comparisons.

All calculations are performed in the arithmetic and logical unit (ALU) of the computer. ALU also does comparisons and takes decision. Whenever any calculation has to be done, the control unit transfers the required data from the storage unit to the ALU.

(iii) Memory unit

Memory unit is the part of computer which holds data for processing and other information. It also called as main memory. It is a temporary storage (volatile). The values stored in the processor with the help of the many register like accumulator, base register. This registers store the value at the time of calculation.

(iv) Secondary storage devices

The secondary storage is used like an archive. It may store several programs, documents, database, etc. The program that you want to run on the computer is first transferred to the primary memory before it can run. The secondary memory is slower than the primary memory. Some of the commonly used secondary memory devices are hard disk, compact disk (CD), digital video disk (DVD). It is the permanent storage; the data stored in this storage will not be erased when the power is lost. It is a non-volatile memory.

3. Output unit

These devices used to get the response or result from the processor. Output unit is the communication between the user and the computer. The output unit of a computer provides the information and results of a computation to the outside world. Commonly used output devices are printer, plotter, and monitor.

Following are few of the important output devices which are used in Computer Systems

- Monitors
- Graphic Plotter
- Printer

Monitors

Monitor commonly called as Visual Display Unit (VDU) is the main output device of a computer. It forms images from tiny dots, called pixels, that are arranged in a rectangular form. The sharpness of the image depends upon the no. of the pixels.

There are two kinds of viewing screen used for monitors.

- Cathode-Ray Tube (CRT)
- Flat- Panel Display

Cathode-Ray Tube (CRT) Monitor

In the CRT display is made up of small picture elements called pixels for short. The smaller the pixels, the better the image clarity, or resolution. It takes more than one illuminated pixel to form whole character, such as the letter e in the word help.

A finite number of character can be displayed on a screen at once. The screen can be divided into a series of character boxes - fixed location on the screen where a standard character can be placed.

The most screens are capable of displaying 80 characters of data horizontally and 25 lines vertically. There are some disadvantage of CRT

- Large in Size
- High Power consumption

Fig: CRT Monitor

Flat-Panel Display Monitor

The flat-panel display refers to a class of video devices that have reduced volume, weight and power requirement compare to the CRT. You can hang them on walls or wear them on your wrists. Current uses for flat-panel displays include calculators, videogames, monitors, laptop computer, graphics display.

The flat-panel display are divided into two categories

- **Emissive Displays** - The emissive displays are devices that convert electrical energy into light. Example are plasma panel and LED (Light-Emitting Diodes).

- **Non-Emissive Displays** - The Non-emissive displays use optical effects to convert sunlight or light from some other source into graphics patterns. Example is LCD (Liquid-Crystal Device)

Fig: Flat –Panel Display Monitor

Printers

Printer is the most important output device, which is used to print information on paper.

There are two types of printers

- Impact Printers
- Non-Impact Printers

Impact Printers

The printers that print the characters by striking against the ribbon and onto the paper are called impact printers.

Characteristics of Impact Printers are following

Very low consumable costs
Impact printers are very noisy
Useful for bulk printing due to low cost
There is physical contact with the paper to produce an image

These printers are of two types

Character printers
Line printers

Character Printers:

Character Printers are printers which print one character at a time.

These are of further two types

Dot Matrix Printer (DMP)

Daisy Wheel

Dot Matrix Printer

In the market one of the most popular printer is Dot Matrix Printer because of their ease of printing features and economical price. Each character printed is in form of pattern of Dot's and head consists of a Matrix of Pins of size(5*7, 7*9, 9*7 or 9*9) which comes out to form a character that is why it is called Dot Matrix Printer.

Advantages

- Inexpensive
- Widely Used
- Other language characters can be printed

Fig: Dot Matrix Printer

Daisy Wheel

Head is lying on a wheel and Pins corresponding to characters are like petals of Daisy (flower name) that is why it is called Daisy Wheel Printer. These printers are generally used for word-processing in offices which require a few letters to be send here and there with very nice quality representation.

Advantages

More reliable than DMP's

Better quality

The fonts of character can be easily changed.

Disadvantages

Slower than DMP's

Noisy

Fig: Daisy Wheel Printer

Line Printers

Line printers are printers which print one line at a time.

These are of further two types

- Drum Printer
- Chain Printer

Fig: Line Printer

Drum Printer

This printer is like a drum in shape so it called drum printer. The surface of drum is divided into number of tracks. Total tracks are equal to size of paper i.e for a paper width of 132 characters, Drum will have 132 tracks. A character set is embossed on track. The different characters sets are available in market 48 character set, 64 and 96 characters set. One rotation of drum prints one line. Drum Printers are fast in speed and speed in between 300 to 2000 lines per minute.

Advantages

- Very high speed

Disadvantages

- Very expensive
- Characters fonts cannot be changed

Chain Printer

In this printer chain of character sets are used so it called Chain Printers. A standard character set may have 48, 64, 96 characters.

Advantages

- Character fonts can easily be changed.
- Different languages can be used with the same printer.

Disadvantages

- Noisy
- Do not have the ability to print any shape of characters.

Non-impact Printers

The printers that print the characters without striking against the ribbon and onto the paper are called Non-impact Printers. These printers print a complete page at a time, also called as Page Printers.

These printers are of two types

- Laser Printers
- Inkjet Printers

Characteristics of Non-impact Printers

- Faster than impact printers.
- They are not noisy.
- High quality.
- Support many fonts and different character size.

Laser Printers

These are non-impact page printers. They use laser lights to produces the dots needed to form the characters to be printed on a page.

Advantages

- Very high speed.
- Very high quality output.
- Give good graphics quality.
- Support many fonts and different character size.

Disadvantage

- Expensive.
- Cannot be used to produce multiple copies of a document in a single printing .

Fig: Laser Printer

Inkjet Printers

Inkjet printers are non-impact character printers based on a relatively new technology. They print characters by spraying small drops of ink onto paper. Inkjet printers produce high quality output with presentable features.

They make less noise because no hammering is done and these have many styles of printing modes available. Colour printing is also possible. Some models of Inkjet printers can produce multiple copies of printing also.

Advantages

- High quality printing
- More reliable

Disadvantages

- Expensive as cost per page is high
- Slow as compare to laser printer

Fig: Inkjet Printer

2 Types of computers

The computers come in many sizes and capabilities. It can be classified into the following based on hardware, utility, size and capacity.

1.Based on hardware design	1. Analog 2. Digital 3. Hybrid
2. Based on utility	1. General purpose computer 2. Special purpose computer
3.Based on size and capacity	1. Micro computer 2. Mini computer 3. Mainframe computer 4. Super computer
4.based on mode of use	1. palmtop pc's 2. laptop pc's 3. personal computer 4. work station 5. mainframe system 6. client and server

2.1 Based on hardware design

Based on hardware design, computers are classified as

1. Analog computer
2. Digital computer
3. Hybrid computer

Analog Computer

Fig: analog computer

Analog is Greek word which means similar. So in analog computes, any two quantities are measured by electrical voltage or current. The analog computer operates by measuring instead of counting. The analog computer works on the supply of continuous electrical signals. The display is also continuous and its output is in the form of graphs. Today analog computers are obsolete.

Digital computer

Fig: digital computer

As the name implies, the digital computer handles quantities represented as digits. In digital computer, both numerical and non-numeric information's are represented as string of digits.

In digital computer, the input data is discrete in nature. It is represented by binary notation in the form of 0's and 1's. Digital computer are much faster than analog computers and far more accurate. Digital computer is largely used for business and scientific applications.

Hybrid computer

Fig: Hybrid computer

In hybrid computer, an attempt is made to combine the qualities of both analog and digital computer. In a hybrid computer, the measuring functions are performed by the analog way while control and logic functions are digital in nature. Weather-monitoring system and devices used in intensive care units of the hospitals are examples of hybrid computer.

2.2 Based on utility

Based on utility, the computers can be classified into

1. General purpose computers
2. Special purpose computers

General purpose computers

These are designed and constructed to cater to almost all the needs of the society. They can perform various operations. They are able to perform according to the programs created to meet different needs.

These can be used for a variety of tasks form financial accounting to mathematical calculations, form designing textile prints to controlling machinery. They are also flexible and can be used to work on business and scientific problems.

Special purpose computers

Special purpose computers can be designed to perform specific functions. In such devices, the instructions are permanently pre programmed. The instructions needed to perform the particular task are incorporated into the internal memory of the computer. Some of the special purpose computer are aircraft control system, electronic voting machines etc.

2.3 Based on size and capacity

Based on size and capacity, computers are classified into the following:

1. Micro computer
2. Mini computer
3. Mainframe computer
4. Super computer

Micro computers

In view of its small size and the use of Microprocessor, this computer is called Micro computer. A microprocessor is a processor whose components namely Input, Output and CPU are on a single integrated-circuit chip. It is a low-cost and the word length of a microcomputer lies in the range of 8 to 32 bits.

Fig:Micro computer

They are normally single-microprocessor, single-user systems designed for performing basic operations like general purpose calculations, industrial control, instrumentation, educational, training, small business applications, home applications, commercial equipment control, watches, fuel injection of a car, office automation, playing games etc.

Mini computer

Minicomputer is larger than the micro computers and is more powerful in terms of processing power. Minicomputer is mainly multiprocessor systems where many users simultaneously work on the systems. Minicomputer possesses greater storage capacity and larger memories as compared to microcomputer. Their word length is 32 bits.

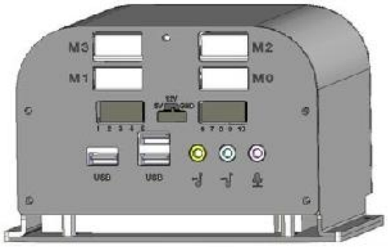


Fig:Mini computer

Mainframe computers

Mainframe computer are larger, faster and more expensive than other general purpose computers. These are used to handle huge volumes of data. Their word length may be 48 to 64 bit memory capacity. These computers even possess and work with more than one processor at the same time.

Fig:Mainframe computer

Super computers

Super computers are the most powerful of all computers. They have a high processing speed. Their processing speed lies in the range of 400 to 10,000 MIPS. Word length 64 to 96 bit. It is specially designed to maximize the number of FLOPS. Their FLOPS rating is usually more than 1 gigaflop per second.

Fig:Super computer

2.4 Based on mode of use

However with the rapidly developing technology, such classifications are no more relevant. After a few months of introduction of a new computer in the marker, the new models of computers are rapidly introduced with less cost and higher performance than the earliest. Thus, today's computers are classified based on their mode of use.

Palmtop pc's

The palmtop computers accept handwritten inputs using an electronic pen, which can be used to write on a palmtop's screen. These have small disk storage and can be connected to a wireless network. One has to train the system on the user's handwriting before it can be used. A palmtop computer has also facilities to be used as a mobile phone, fax and E-mail machine.

Fig:World first palmtop pc's

Laptop pc's

Laptop pc's are portable computers with less weight and small enough to be kept on a lap. They use a keyboard, Flat screen liquid crystal display, and a Pentium or power pc processor. Color displays are also available. It is a battery powered personal computer generally smaller than a briefcase that can easily be transported and conveniently used in temporary space. Laptops come with hard disk, CD ROM and floppy disk drives.

Fig:Laptop pc's

Personal computers (PC's)

The name itself implies, the personal computers were mainly designed to meet the personal computing requirements of users at working place or in home. It is a non-portable and general purpose computer which can easily fit on a normal size office table.

Fig:Personal computer

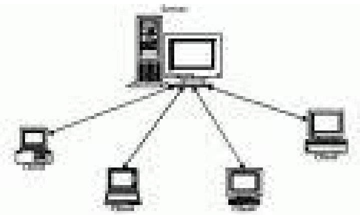
Work stations

It is a powerful desktop computer designed to meet the computing needs of users or clients, with better processing, high storage capacity and with efficient and effective graphics display facility. The workstation looks similar to a personal computer and can be used by only one person at a time through the local area network.

Fig:Work station

Clients and servers

With the increased gain of computer networks, it is possible to interconnect several computer for communication over the networks to share the several resources or services among the multiple users.



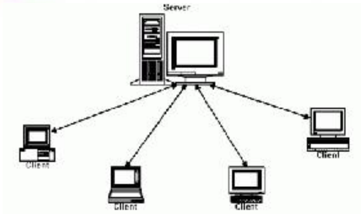


Fig:Client server

Computers come in many different forms. They range from massive, multipurpose mainframes and supercomputers to desktop-size personal computers. Between these extremes is a vast middle ground of minicomputers and workstations. Large minicomputers approach mainframes in computing power, whereas workstations are powerful personal computers.

Mainframes and large minicomputers are used by many businesses, universities, hospitals and government agencies to carry out sophisticated scientific and business calculations. These computers are expensive and many require a sizeable staff of supporting personnel and a special, carefully controlled environment.

Personal computers, on the other hand, are small and inexpensive. In fact, portable, battery-powered "laptop" computers weighing less cost. Personal computers are used extensively in most schools and businesses and they are rapidly becoming common household items. Many students use personal computers when learning program with C.

The small size and low cost, modern personal computers approach minicomputers in computing power. They are now used for many applications that formerly required larger, more expensive computers. Their performance continues to improve dramatically as their cost continues to drop. The design of a personal computer permits a high level of interaction between the user and the computer. Most applications are specifically designed to take advantage of this feature thus providing the skilled user with a wide variety of creative tools to write, draw or carry out numerical computations.

3. Number systems

Introduction

A number is required for counting or to express the amount of some quantity. It consists of group of symbols called digits, which are arranged in a definite manner. There can be many groups of symbols called digits, which are arranged in a definite manner. There can be many ways in which the digits can be arranged to form a number. This gives rise to what we call a number system. Each such system deals with a different set of digits to form a number belonging to that system.

The most widely adopted system is the decimal number system which has ten digits (0,1,2,3 …9), the octal system has eight digits (0,1,2,3…, 7), the hexadecimal system has sixteen digits (0,1,2,3… ,9,A,B,C,D,E,F), the binary number system has only two (0,1), symbols.

The number of digits in a system is called 'radix' or 'base'. So that decimal system may be called as radix-10 system, the binary system as radix-2 system.

NUMBER SYSTEMS

Binary	Decimal	Octal	Hexadecimal
0000	00	0	0
0001	01	1	1
0010	02	2	2
0011	03	3	3
0100	04	4	4
0101	05	5	5
0110	06	6	6
0111	07	7	7
1000	08	10	8
1001	09	11	9
1010	10	12	A
1011	11	13	B
1100	12	14	C
1101	13	15	D
1110	14	16	E
1111	15	17	F

3.1 Conversion of number system

Usually the numbers are expressed in decimal systems, because we are using decimal number system in our daily life since decades. Thus any number from one number system can be represented to any other number system. Since the input and the output values are to be decimal, and the computer uses other number systems the computer professionals are required to convert decimal number system to other number and in other number system to decimal.

Different number system conversions are followed.

1. Decimal to binary
2. Binary to decimal
3. Decimal to octal
4. Decimal to Hexadecimal
5. Octal to decimal
6. Octal to binary
7. Binary to octal
8. Hexadecimal to binary
9. Binary to hexadecimal
10. Hexadecimal to decimal
11. Octal to hexadecimal
12. Hexadecimal to Octal

1. Conversions of Decimal to Binary- The method that is used for converting of decimals into binary is known as the remainder method. We use the following steps in getting the binary number-

(a) Divide the decimal number by 2.

(b) Write the remainder (which is either 0 or 1) at the right most position.

(c) Repeat the process of dividing by 2 until the quotient is 0 and keep writing the remainder after each step of division.

(d) Write the remainders in reverse order.

Example- Convert $(45)_{10}$ into binary number system.

2	45	**Remainder**
2	22	1
2	11	0
2	5	1
2	2	1
2	1	0
	0	1

Thus $(45)_{10} = (101101)_2$

Note- In every number system-

(a) The first bit from the right is referred as LSB (Least Significant Bit)

(b) The first bit from the left is referred as MSB (Most Significant Bit)

Conversions of Decimal Fractions to Binary Fractions- For converting decimal fractions into binary fractions, we use multiplication. Instead of looking for a remainder we look for an integer. The following steps are used in getting the binary fractions-

(a) Multiply the decimal fraction by 2.

(b) If a non-zero integer is generated, record the non-zero integer otherwise record 0.

(c) Remove the non-zero integer and repeat the above steps till the fraction value becomes 0.

(d) Write down the number according to the occurrence.

Example- Find the binary equivalent of $(0.75)_{10}$.

	Number (to be recorded)
0.75 × 2 = 1.50	1
0.50 × 2 = 1.00	1

Thus $(0.75)_{10}= (0.11)_2$.

Moreover, we can write $(45.75)_{10}= (101101.11)_2$.

Remark- If the conversion is not ended and still continuing; we write the approximation in 16 bits.

Example- Find the binary equivalent of $(0.9)_{10}$.

	Number (to be recorded)
0.9 × 2 = 1.8	1
0.8 × 2 = 1.6	1
0.6 × 2 = 1.2	1
0.2 × 2 = 0.4	0
0.4 × 2 = 0.8	0

0.8 × 2 = 1.6 1

0.6 × 2 = 1.2 1

0.2 × 2 = 0.4 0

0.4 × 2 = 0.8 0

0.8 × 2 = 1.6 1

0.6 × 2 = 1.2 1

0.2 × 2 = 0.4 0

0.4 × 2 = 0.8 0

0.8 × 2 = 1.6 1

0.6 × 2 = 1.2 1

0.2 × 2 = 0.4 0

0.4 × 2 = 0.8 0

0.8 × 2 = 1.6 1

Thus $(0.9)_{10} = (0.111001100110011001)_2$.

2. Conversions of Binary to Decimal – In converting binary to decimal, we use the following steps-

(a) Write the weight of each bit.

(b) Get the weighted value by multiplying the weighted position with the respective bit.

(c) Add all the weighted value to get the decimal number.

Example- Convert $(101101)_2$ into decimal number system.

Binary Number	1	0	1	1	0	1
Wt. of each Bit	2^5	2^4	2^3	2^2	2^1	2^0
Weighted Value	1×2^5	0×2^4	1×2^3	1×2^2	0×2	1×2^0
Solved Multiplication	32	0	8	4	0	1

Thus $(101101)_2 = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2 + 1 \times 2^0$.

$= 32 + 0 + 8 + 4 + 0 + 1$

$= 45$

Conversions of Binary Fractions to Decimal Fractions – The conversions of binary fractions to the decimal fractions is similar to conversion of binary numbers to decimal numbers. Here, instead of a decimal point we have a binary point. The exponential expressions (or weight of the bits) of each fractional placeholder is 2^{-1}, 2^{-2}………

Example- Convert $(101101.11)_2$ into decimal number system.

Binary Number	1	0	1	1	0	1	1	1
Wt. of each Bit	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}
Weighted Value	1×2^5	0×2^4	1×2^3	1×2^2	0×2	1×2^0	1×2^{-1}	1×2^{-2}
Solved Multiplication	32	0	8	4	0	1	0.5	0.25

Thus $(101101.11)_2$= 32 + 0 + 8 + 4 + 0 + 1 + 0.5 + 0.25 = 45.75

3. Conversion of Decimal to Octal- In converting decimal to octal, we follow the same process of converting decimal to binary. Instead of dividing the number by 2, we divide the number by 8.

Example- Convert $(45)_{10}$ into octal number system.

8	45	**Remainder**
8	5	5
8	0	5

Thus $(45)_{10}$= $(55)_8$.

Conversions of Decimal Fractions to Octal Fractions – We follow the same steps of conversions of decimal fractions to binary fractions. Here we multiply the fraction by 8 instead of 2.

Example- Find the octal equivalent of $(0.75)_{10}$.

	Number (to be recorded)
0.75 × 8 = 6.00	6

Thus $(0.75)_{10}$= $(0.6)_8$.

And $(45.75)_{10}$= $(55.6)_8$.

4. Conversion of Decimal to Hexadecimal – We divide by 16 instead of 2 or 8. If the remainder is in between 10 to 16, then the number is represented by A to F respectively.

Example- Convert $(45)_{10}$ into hexadecimal.

16	45	**Remainder**
16	2	D
16	0	2

Thus $(45)_{10}$= $(2D)_{16}$.

Conversions of Decimal Fractions to Hexadecimal Fractions – Here we multiply the fraction by 16 instead of 2 or 8. If the non-zero integer is in between 10 to 16, then the number is represented by A to F respectively.

Example- Find the hexadecimal equivalent of $(0.75)_{10}$.

Number (to be recorded)

0.75 × 16 = 12.00 C (12 = C) Thus $(0.75)_{10}$=

$(0.C)_{16}$.

And $(45.75)_{10}$= $(2D.C)_{16}$.

5. Conversions of Octal to Decimal- We follow the same steps of conversion of binary to decimal. The only difference is that here weight of n^{th} bit is 8^{n-1} instead of 2^{n-1}.

Example- Convert $(55)_8$ into decimal number system.

Octal Number	5	5
Wt. of each Bit	8^1	8^0
Weighted Value	5× 8	5×8^0
Solved Multiplication	40	5

Thus $(55)_8$ = 40 + 5.

= 45

Conversions of Octal Fractions to Decimal Fractions- The weight of the bit of the fraction placeholder is 8^{-1}, 8^{-2}.......... We follow the same steps of conversion of binary fractions to decimal fractions.

Example- Convert $(55.6)_8$ into decimal number system.

Octal Number	5	5	6
Wt. of each Bit	8^1	8^0	8^{-1}
Weighted Value	5 × 8	5×8^0	6×8^{-1}
Solved Multiplication	40	5	0.75

Thus $(55.6)_8$= 40 + 5 + 0.75 = 45.75

6. **Conversions of Octal to Binary-** We use the following steps in converting octal to binary-
   (a) Convert each octal digit into 3-bit binary equivalent.
   (b) Combine the 3-bit section by removing the spaces to get the binary number.

Example- Convert $(55)_8$ into binary.

Octal Number	5	5
Binary Number	101	101

Thus $(55)_8 = (101101)_2$.

Example- Convert $(456)_8$ into binary.

Octal Number	4	5	6
Binary Number	100	101	110

Thus $(456)_8 = (100101110)_2$.

Conversions of Octal Fractions to Binary Fractions- We follow the same steps of conversion of octal to binary.

Example- Convert $(55.6)_8$ into binary.

Octal Number	5	5	6
Binary Number	101	101	110

Thus $(55.6)_8 = (101101.11)_2$.

7.Conversions of Binary to Octal- We use the following steps in converting binary to octal-

(a) Break the number into 3-bit sections starting from LSB to MSB.

(b) If we do not have sufficient bits in grouping of 3-bits, we add zeros to the left of MSB so that all the groups have proper 3-bit number.

(c) Write the 3-bit binary number to its octal equivalent.

Example- Convert $(101101)_2$ into octal.

Binary Number	101	101
Octal Number	5	5

Thus $(101101)_2 = (55)_8$.

Example- Convert $(1101101)_2$ into octal.

Binary Number	001	101	101
Octal Number	1	5	5

Thus $(1101101)_2= (155)_8$.

Conversions of Binary Fractions to Octal Fractions- We use the following steps in converting binary fractions to octal fractions-

(d) Break the fraction into 3-bit sections starting from MSB to LSB.

(e) In order to get a complete grouping of 3 bits, we add trailing zeros in LSB.

(f) Write the 3-bit binary number to its octal equivalent.

Example- Convert $(101101.11)_2$ into octal.

Binary Number	101	101	110
Octal Number	5	5	6

Thus $(101101)_2 = (55.6)_8$.

8. Conversions of Hexadecimal to Binary-We use the following steps-

(a) Convert each hexadecimal digit to its 4-bit binary equivalent.

(b) Combine all the binary numbers.

Example- Convert $(2D)_{16}$ into binary.

Hexadecimal Number	2	D(=13)
Binary Number	0010	1101

Thus $(2D)_{16}$ = $(00101101)_2$= $(101101)_2$.

Conversions of Hexadecimal Fractions to Binary Fractions -We use the same steps of hexadecimal to binary conversion.

Example- Convert $(2D.C)_{16}$ into binary.

Hexadecimal Number	2	D(=13)	C(=12)
Binary Number	0010	1101	1100

Thus $(2D)_{16}$ = $(00101101.1100)_2$= $(101101.11)_2$.

9. Conversions of Binary to Hexadecimal- We convert binary to hexadecimal in the similar manner as we have converted binary to octal. The only difference is that here, we form the group of 4 – bits.

Example- Convert $(101101)_2$ into hexadecimal.

Binary Number	0010	1101
Decimal Number	2	13
Hexadecimal Number	2	D

Thus $(101101)_2$ = $(2D)_{16}$.

Conversions of Binary Fractions to Hexadecimal Fractions - We convert binary fractions to hexadecimal fractions in the similar manner as we have converted binary fractions to octal fractions. The only difference is that here we form the group of 4 – bits.

Example- Convert $(101101.11)_2$ into hexadecimal.

Binary Number	0010	1101	1100
Decimal Number	2	13	12
Hexadecimal Number	2	D	C

Thus $(101101.11)_2$ = $(2D.C)_{16}$.

10. Conversions of Hexadecimal to Decimal- We do the conversion of hexadecimal to decimal as we have done the conversion of binary to decimal. Here weight of n^{th} bit is 16^{n-1} instead of 2^{n-1}.

Example- Convert $(2D)_{16}$ into decimal.

Hexadecimal Number	2	D(=13)
Wt. of each Bit	16^1	16^0
Weighted Value	2 × 16	13 × 16^0
Solved Multiplication	32	13

Thus $(2D)_{16}$= 32 + 13 = 45.

Conversions of Hexadecimal Fractions to Decimal Fractions- We do the conversion of hexadecimal fractions to decimal fractions in the similar manner as we have done the conversion of binary fractions to decimal fractions. Here weight of bit is 16^{-1}, 16^{-2}…….

Example- Convert $(2D.C)_{16}$ into decimal.

Hexadecimal Number	2	D(=13)	C(=12)
Wt. of each Bit	16^1	16^0	16^{-1}
Weighted Value	2 × 16	13 × 16^0	13 × 16^{-1}
Solved Multiplication	32	13	0.75

Thus $(2D.C)_{16}$= 32 + 13 + 0.75 = 45.75.

11.Conversions of Octal to Hexadecimal- The conversion involves the following steps-

(a) Convert each octal digit to 3 –bit binary form.

(b) Combine all the 3-bit binary numbers.

(c) Group them in 4-bit binary form by starting from MSB to LSB.

(d) Convert these 4-bit blocks into their hexadecimal symbols.

Example- Convert $(55)_8$ into hexadecimal.

Octal Number	5	5
Binary Number	101	101

Combining the 3-bit binary block, we have 101101.

Grouping them in 4 bit binary form-

Binary Number	0010	1101
Hexadecimal Symbol	2	D

Thus $(55)_8 = (2D)_{16}$.

Conversions of Octal Fractions to Hexadecimal Fractions- The method of conversion is based on the same procedure that we have discussed in conversions of octal to hexadecimal.

Example- Convert $(55.6)_8$ into hexadecimal.

Octal Number	5	5	6
Binary Number	101	101	110

Combining the 3-bit binary block, we have 101101.110.

Grouping them in 4 bit binary form-

Binary Number	0010	1101	1100
Hexadecimal Symbol	2	D	C

Thus $(55)_8 = (2D.C)_{16}$.

12.Conversions of Hexadecimal to Octal- We convert each hexadecimal digit in binary. Combine all the binary numbers. Again group them into 3-bit form. Convert the 3-bit block in octal.

Example- Convert $(2D)_{16}$ into octal.

Hexadecimal Number	2	D(=13)
Binary Number	0010	1101

Combining the binary number, we get 00101101=101101
Grouping the binary number in 3-bit

Binary Number	101	101
Octal Number	5	5

Thus $(2D)_{16} = (55)_8$.

Conversions of Hexadecimal Fractions to Octal Fractions – We follow the same steps of hexadecimal to octal conversion.

Example- Convert $(2D.C)_{16}$ into octal.

Hexadecimal Number	2	D(=13)	C(=12)
Binary Number	0010	1101	1100

Combining the binary number, we get 00101101.1100=101101.11
Grouping the binary number in 3-bit

Binary Number	101	101	110
Octal Number	5	5	6

Thus $(2D.C)_{16} = (55.6)_8$.

Expected Learning Outcome:

Students will acquire knowledge on Number Systems because number systems are the base of computers. All the Arithmetic calculations are the base of Number systems only. They will get knowledge to convert decimal to binary, octal and hexa decimal and viceversa.

Exercises

Questions:

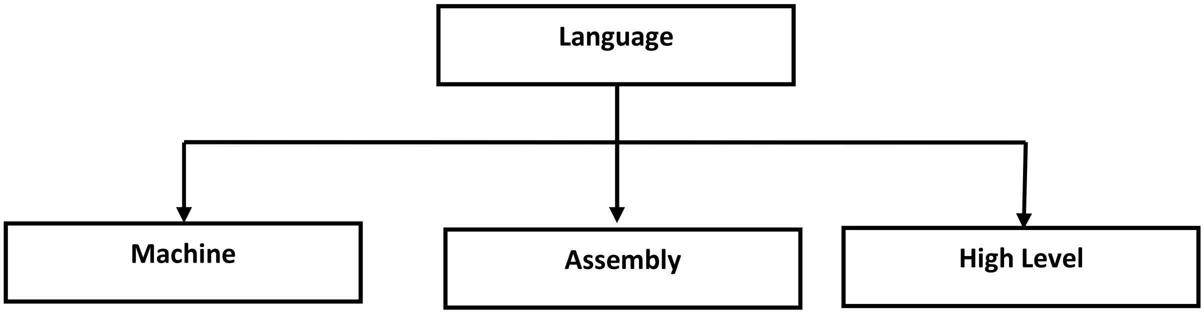
A 1.what are the types of Number System?
2. What is base?

B 1. What is Hexadecimal Number System?
2. What is Binary number System?

C 1. Discuss in detail about the classification of Number System.
2.Explain about the Number System Conversion types with Suitable Example.

5 Computer Languages

Language is the communication medium, the computer languages are used to communicate with the computer. These are the communication media between the user and the computer. The computer languages are formed with the group of instruction often called as programs. The computer languages are divided as follows.



Classification of languages

5.1 Machine languages

The machine language is formed with the help of binary digits. This is also called as the low level language. This language is the collection of instruction using binary digits 0's and 1's. There are as named as the machine can directly understand the programs.

Advantages of machine language:

1. The computer can understand instruction immediately.
2. No translation is needed

Disadvantages of machine language:

1. Machine dependent
2. Writing a Programming is very difficult
3. Very difficult to find an error.

Example: add 4 and 9

```
            0100
            1001
      _____________
            1101
      ______________
```

5.2 Assembly Language

It was developed to overcome some of the many inconveniences of machine language. This is another low level but a very important language in which operation codes and operands are given in the form of alphanumeric symbols instead of 0's and l's. These alphanumeric symbols will be known as mnemonic codes and can have maximum up to 5 letter combination e.g. ADD for addition, SUB for subtraction, START, LABEL etc. Because of this feature it is also known as 'Symbolic Programming Language'.

This language is also very difficult and needs a lot of practice to master it because very small. English support is given to this language. The language mainly helps in compiler orientations. The instructions of the Assembly language will also be converted to machine codes by language translator to be executed by the computer.

Advantages of Assembly Language

i) It is easier to understand and use as compared to machine language.

ii) It is easy to locate and correct errors.

iii) It is modified easily.

Disadvantages of Assembly Language

i) Like machine language it is also machine dependent.

ii) Since it is machine dependent therefore programmer should have the knowledge of the hardware also.

Example: addition of 4 and 9

Say A=4 B=9

```
MVI     A       move A value to Accumulator

ADD     B       add B with Accumulator

STA     C       Store Accumulator contents to C

HLT             Halt the program.
```

5.3 . High Level Languages

High level computer languages give formats close to English language and the purpose of developing high level languages is to enable people to write programs easily and in their own native language environment (English). High-level languages are basically symbolic languages that use English words and/or mathematical symbols rather than mnemonic codes. Each instruction in the high level language is translated into many machine language instructions thus showing one-to-many translation.

Types of High Level Languages

Many languages have been developed for achieving different variety of tasks, some are fairly specialized others are quite general purpose.

These are categorized according to their use as

- **Algebraic Formula-Type Processing.**
- **Business Data Processing.**
- **String and List Processing.**
- **Object Oriented Programming Language.**
- **Visual programming language.**

1. Algebraic Formula-Type Processing:

These languages are oriented towards the computational procedures for solving mathematical and statistical problem

Examples are

- **BASIC (Beginners All Purpose Symbolic Instruction Code).**
- **FORTRAN (Formula Translation).**
- **PL/I (Programming Language, Version 1).**
- **ALGOL (Algorithmic Language).**
- **APL (A Programming Language).**

2. Business Data Processing:

These languages emphasize their capabilities for maintaining data processing procedures and files handling problems. Examples are:

- **COBOL (Common Business Oriented Language).**
- **RPG (Report Program Generator**

3. String and List Processing: These are used for string manipulation including search for patterns, inserting and deleting characters. Examples are:

- **LISP (List Processing).**
- **Prolog (Program in Logic).**

4. Object Oriented Programming Language:

In OOP, the computer program is divided into objects. Examples are:

- **C++**
- **Java**

5. Visual programming language:

These are designed for building Windows-based applications Examples are:

- **Visual Basic**
- **Visual Java**
- **Visual C**

Advantages of High Level Language

Following are the advantages of a high level language:

i) It is easier to learn.
ii) They require less time to write.
iii) They are easier to maintain.
iv) Problem oriented rather than 'machine' based.
v) Program written in a high-level language can be translated into many machine language and therefore can run on any computer for which there exists an appropriate translator.
vi) It is independent of the machine on which it is used i.e., Programs developed in high level language can be run on any Computer.

vii) User-friendly.
viii) Similar to English with vocabulary of words and symbols.

Disadvantages of High Level Language

i) A high-level language has to be translated into the machine language by a translator and thus a price in computer time is paid.

ii) The object code generated by a translator might be inefficient Compared to an equivalent assembly language program.

Further Reading: List the chapters in Reference books/ web resources

http://www.ustudy.in

Summary:

The Types of Computer Languages elaborated about its importance and its advantages and disadvantages.

Expected Learning Outcome:

Students will acquire knowledge about the Types of computer languages.

Exercises

Questions:

A 1.Machine Language converts each bit into
b) byte b)int c) 0 and 1 d) all the three
……………………………
2. What the expansion of FORTRAN……………………………

3. What the expansion of LISP ……………………………

B 1. What are the types of Computer Languages?
2. How High level languages are classified?
3.What is Assembly Level Language?
4. What are advantages and disadvantages of Low level languages?
5. What are advantages and disadvantages of Assembly level languages?

C 1.Explain about the computer languages and its advantages and disadvantages in detail.

6.1 Algorithm

It is defined as a sequence of instructions designed in such a way that if the instructions are executed in the specified sequence, the desired results will be obtained. The algorithm should precise and unambiguous in practice and the results should obtain after a finite number of steps. An algorithm is defined if any problem whose solution can be expressed in a list of executable instruction.

Characteristics of algorithms

In order to qualify algorithms is a sequence of instructions, they must possess the following characteristics:

i) In the algorithms, each and every instruction should be precise.
ii) Each and every instruction should be unambiguous
iii) Ensure that the algorithm will ultimately terminate.
iv) The algorithm should be written in sequence.
v) It looks like normal English
vi) The desired result should be obtained only after the algorithm terminates.

Qualities of a good algorithm

There are so many methods or logics available to solve the problem individually. All of those methods logics may not be good, for given problem; there may be so many algorithms not of all equality. The following are the primary factors that are often used to judge the quality of the algorithm.

s.no	Factors	Quality of good algorithm
1	Time	To execute a program, the computer system takes some amount of time. The lesser is the time required, the better is the algorithm.
2	Memory	Execute the program with in limited memory
3	Accuracy	Multiple algorithms may provide suitable or correct solutions to a given problem, some of these may provide more accurate results than others, such algorithms may be suitable.
4	Sequence	The procedure of an algorithm must form in a sequence and some of the instructions of an algorithm may be repeated in number of times or until a particular condition is met.
5	Generalized	The designed algorithm must solve a single isolated problem and more often algorithms are desired to handle a range of input data to meet this criteria, so the algorithms must be generalized.

Representation of algorithms

The algorithms can be represented in several ways. Generally the programmers follows one of the following ways to represent an algorithms:

i) Normal English
ii) Flowchart
iii) Pseudo code
iv) Decision table
v) Program

1	Normal English	The algorithm can be easily represented in step by step sequential order in normal English, such algorithms are easy to understand, write and read.
2	Flow chart	The flowchart is a pictorial representation of algorithms, the sequential steps in an algorithm can be represented as a flowchart using the standard symbols.
3	Pseudo code	The flowcharts are only one of the possible format decision tools. Whereas pseudo code is also a formal design tools and is very well with the rules of structured design and programming.
4	Decision table	A decision table helps a lot in designing a specific segment of design. It provides another way to look at a complex, nested selection to help clarify the conditions to be tested and how those conditions should be nested to arrive at the proper actions.
5	Program	The algorithms can be implemented as a program using any high level language that becomes a program.

A sample algorithm

Write an algorithm to calculate and print the total number of students selected in various group, using the following criteria. In a school there are 100 student appeared for examination. Based upon the student marks who secure greater than 70% as first class, greater than 60% as second class, and greater than 50% as third class and remaining are fail.

Algorithm

Step 1: start the program

Step 2: read number of students, name of the student, marks.

Step 3: calculate the total marks obtained by each student.

Step 4: check student mark is greater than 70 or not. If greater than 70 then “first class”.

Step 5: check student mark is greater than 60 and less than 70or not. If so then, “second class”.

Step 6: check student mark is greater than 50 and less than 60 or not. If so then, “third class”.

Step 7: print the result of the student with their names and marks.

Step 8: stop the program.

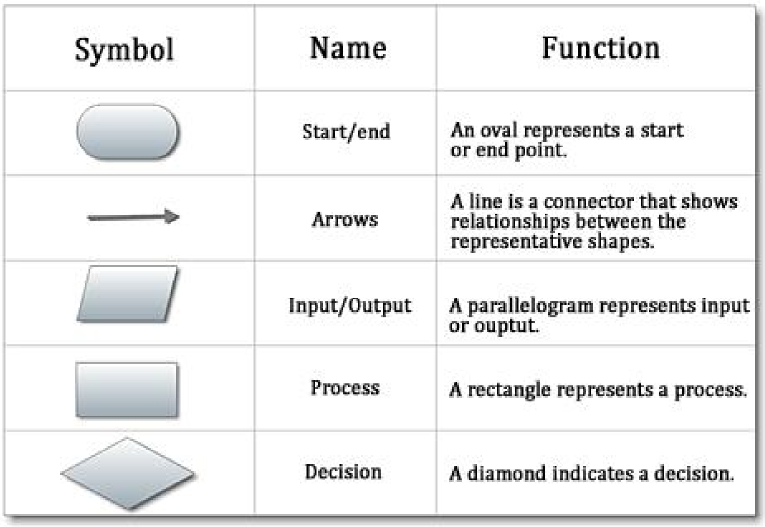
6.2 Flowchart

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

Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields.

Basic Flowchart Symbols

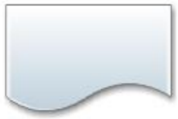
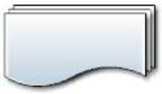
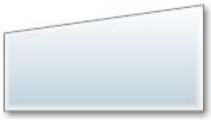
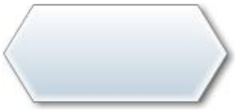
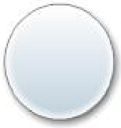
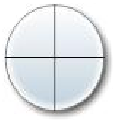
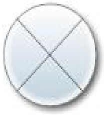
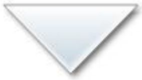
A flowchart is the visualization of a system or process consisting of five basic objects:

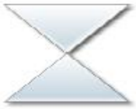
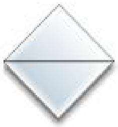
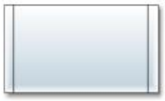
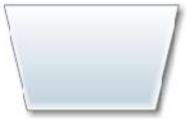
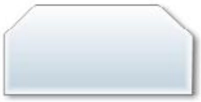
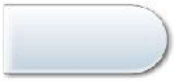
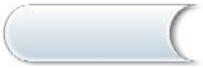


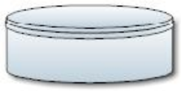
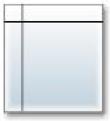
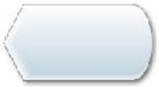
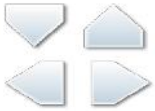
Symbol	Name	Function
⬭	Start/end	An oval represents a start or end point.
→	Arrows	A line is a connector that shows relationships between the representative shapes.
▱	Input/Output	A parallelogram represents input or ouptut.
▭	Process	A rectangle represents a process.
◇	Decision	A diamond indicates a decision.

More Flowchart Symbols

Flowcharts use special shapes to represent different types of actions or steps in a process. Lines and arrows show the sequence of the steps, and the relationships among them.

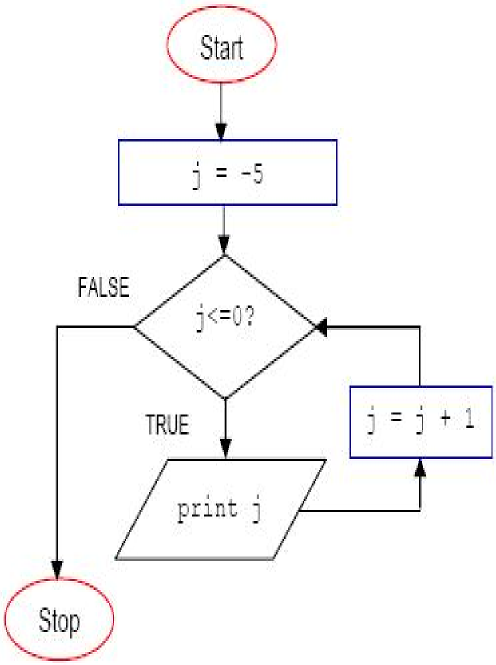
Symbol	Name	Function
	**Document Symbol**	A printed document or report.
	**Multiple Documents Symbol**	Represents multiple documents in the process.
	**Manual Input Symbol**	Represents a step where a user is prompted to enter information manually.
	**Preparation Symbol**	Represents a set-up to another step in the process.
	**Connector Symbol**	Indicates that the flow continues where a matching symbol (containing the same letter) has been placed.
	**Or Symbol**	Indicates that the process flow continues in more than two branches.
	**Summoning Junction Symbol**	Indicates a point in the flowchart where multiple branches converge back into a single process.
	**Merge Symbol**	Indicates a step where two or more sub-lists or sub-processes become one.

Symbol	Name	Function
	**Collate Symbol**	Indicates a step that orders information into a standard format.
	**Sort Symbol**	Indicates a step that organizes a list of items into a sequence or sets based on some pre-determined criteria.
	**Subroutine Symbol**	Indicates a sequence of actions that perform a specific task embedded within a larger process. This sequence of actions could be described in more detail on a separate flowchart.
	**Manual Loop Symbol**	Indicates a sequence of commands that will continue to repeat until stopped manually.
	**Loop Limit Symbol**	Indicates the point at which a loop should stop.
	**Delay Symbol**	Indicates a delay in the process.
	**Data Storage or Stored Data Symbol**	Indicates a step where data gets stored.

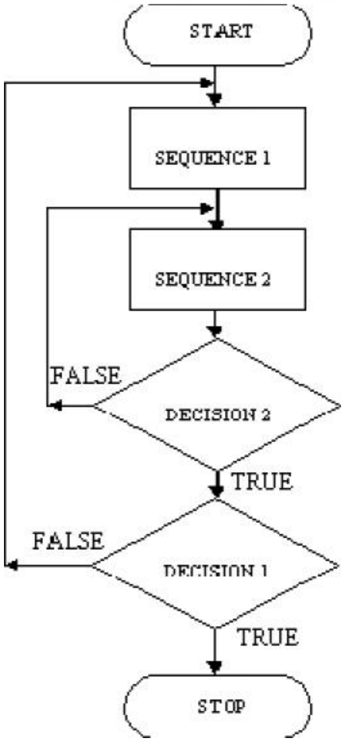
Symbol	Name	Function
	**Database Symbol**	Indicates a list of information with a standard structure that allows for searching and sorting.
	**Internal Storage Symbol**	Indicates that information was stored in memory during a program, used in software design flowcharts.
	**Display Symbol**	Indicates a step that displays information.
	**Off Page**	Indicates that the process continues off page.

FLOWCHART FOR DECISION MAKING STATEMENTS:

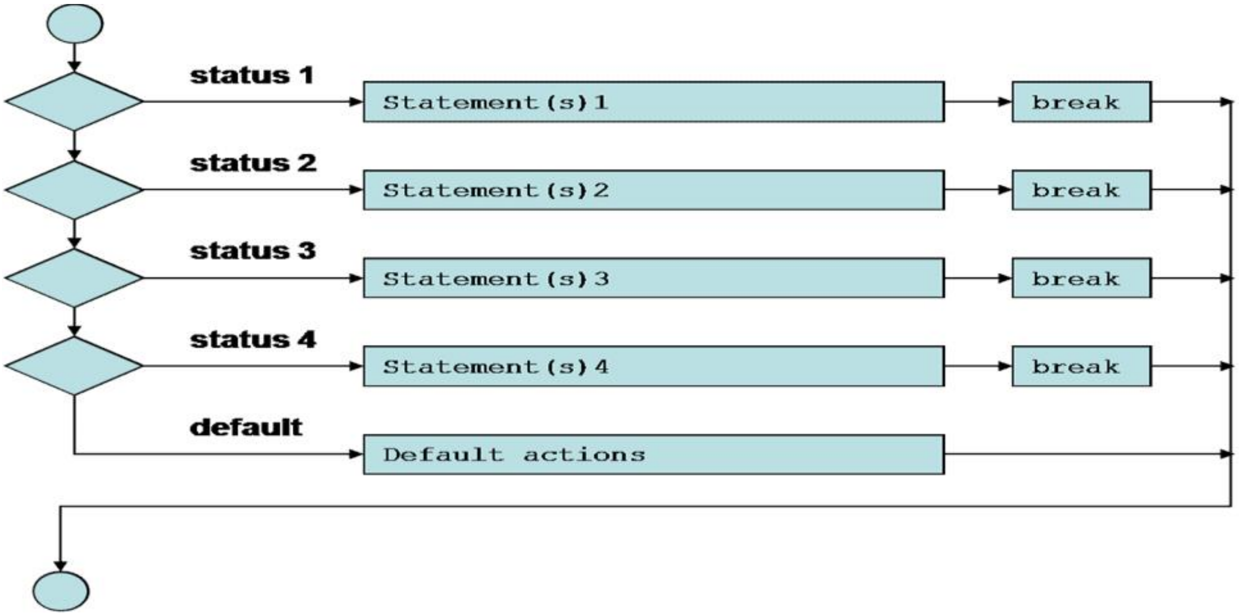
1. A simple flow chart showing the symbols for decision making statements(Simple IF):



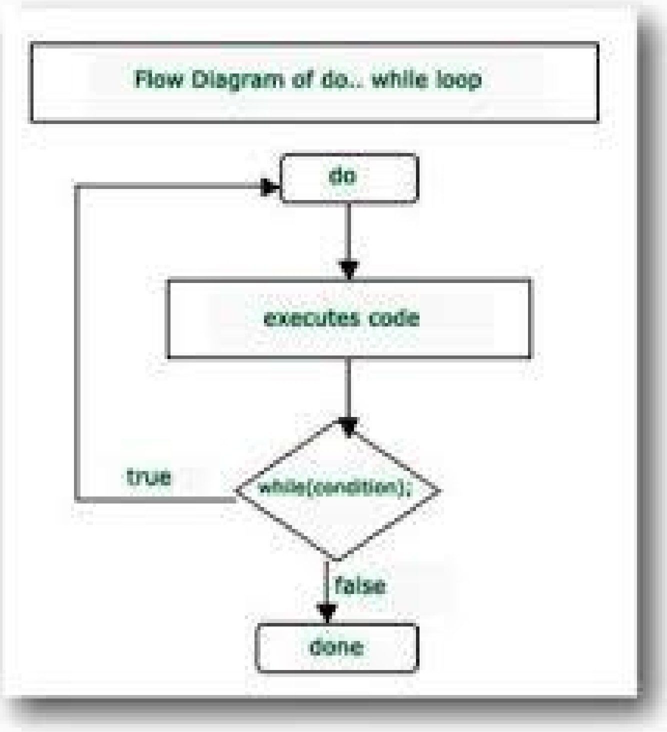
2. A simple flow chart showing the symbols for decision making statements(Nested IF):



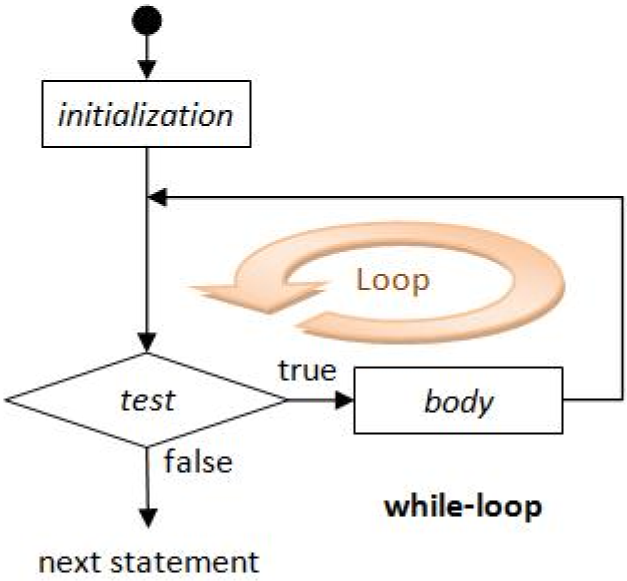
3. A simple flow chart showing the symbols for decision making statements(SWITCH):



4. A simple flow chart showing the symbols for Looping statements(Do-While):



5. A simple flow chart showing the symbols for Looping statements(While):



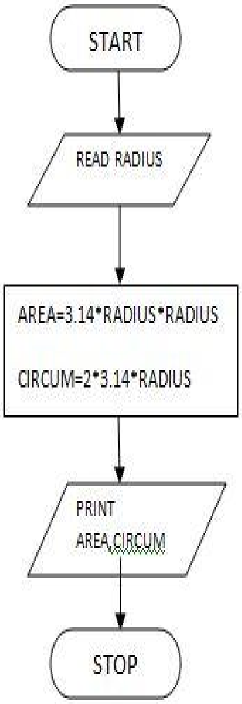
EXAMPLES:

1. ALGORITHM & FLOWCHART TO FIND THE AREA OF A CIRCLE:

Algorithm:

```
Step 1: Include header files.
Step 2: Read the value of radius.
Step 3: Calculate the area and circumference of a circle using the formula.
            A=3.14*r*r;
            C=2*3.14*r;
Step 4: Print the values.
Step 5: Exit.
```

Flowchart:



2. ALGORITHM & FLOWCHART TO FIND THE GIVEN NUMBER IS EVEN OR ODD.

Algorithm:

Step 1: Include Header files.

Step 2: Read the value of a number.

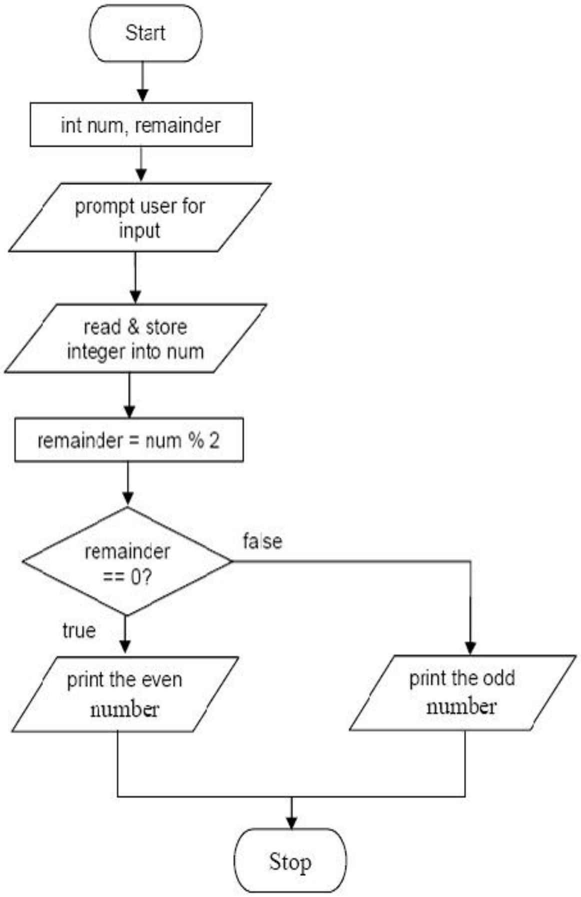
Step 3: Check the given number by using if (n%2==0)

Step 4: if its remainder is zero, Print it is even number.

Step 5: If its remainder is one, Print it is odd number.

Step 6: Exit

Flowchart:



3. ALGORITHM AND FLOWCHART TO PRINT THE NUMBERS FROM 1 to 10.

Algorithm:

Step 1: Include Header files.

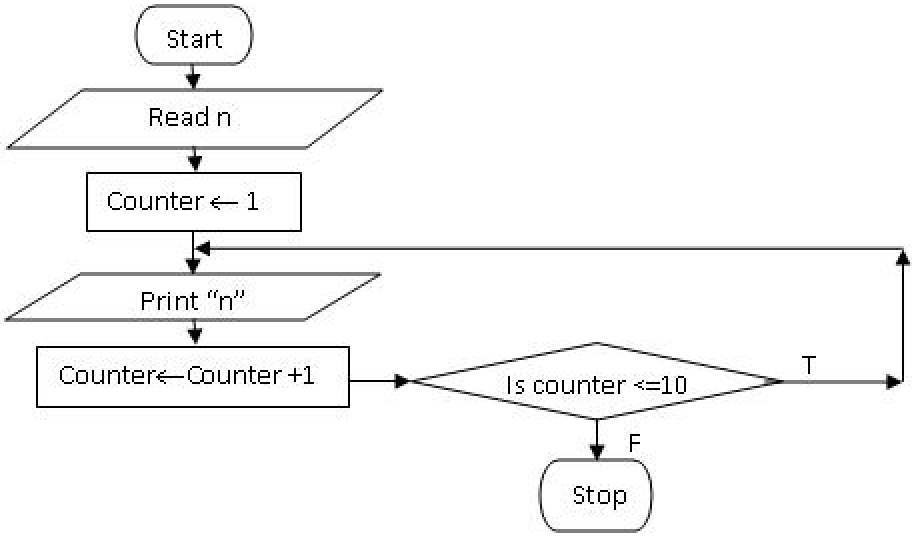
Step 2: Read the value of n.

Step 3: Initialize the value of counter by 1 using counter variable.

Step 4: Print the values upto the value of n by checking with if statement if(n<10).

Step 5: Exit.

Flowchart:



Further Reading: List the chapters in Reference books/ web resources

http://www.ustudy.in

Summary:

This content describes the importance of algorithm and flowchart elaborately.

Expected Learning Outcome:

Students will acquire knowledge about how to write algorithm for a program and they can draw flowchart for each programs.

Exercises

Questions:

A 1.A Step by Step procedure to describe a program is

a) Algorithm b) flowchart c) program d) all the three

2. What is the symbol for database ……………………………

3. What is the symbol for connector……………………………

B 1. What is algorithm?

2. Define flowchart.

3. What is the importance of Algorithm?

4. Write an algorithm to find the area of a triangle?

5.Draw a flowchart for printing the biggest among two numbers?

C 1. Explain about importance of algorithm and flowchart. Briefly describe all the symbols to denote in flowchart..

COURSE PLAN

Select a course (a theory subject): **Programming for Problem Solving**

Select a unit: **Module II**

Select a Topic: **Introduction to C Programming**

1. **Objectives**

   To study about the Basic of C-Program language

2. **Outcomes**
3. **Pre-requisites**
4. **Terminology used other than normal known scientific / engg terms and their fundamental explanations / relations**

Plan for the lecture delivery

1. **How to plan for delivery – black board / ppt / animated ppt / (decide which is good for this topic)**

   Power Point Presentation with Online Compiler

2. **How to explain the definition and terms with in it**

   Define the variable, constant, keywords and rules for coming variable, constant keywords and explain the structure of C Program.

3. **How to start the need for any derivation / procedure / experiment / case study**

   Not Applicable

4. **Physical meaning of math equations / calculations**

   Not Applicable

5. **Units and their physical meaning to make them understand practical reality and comparison between different units**

   This unit is the basic for writing a C Programming. Understand this unit will be to write a simple program.

6. **What is the final conclusion?**

   Students will acquire knowledge basic structure of C Language

7. **How to put it in a nut shell**

   Variables, Constant, Keywords, Input / output Statement.

8. **Important points for understanding / memorizing / make it long lasting**
   - Variable is nothing but name to the memory location and the value may be changed during the program execution.
   - Constant is nothing but the value of the variable can't changed during the program execution.
   - Keyword is a reserved word. Can't used this words as variable name.
   - Printf / Scanf are input / output statement in C language.
   - Operator is nothing but it is a symbol to perform various arithmetic Operations.
     - Arithmetic Operator
     - Assignment Operator.
     - Logical Operator
     - Ternary / Conditional Operator
     - Relational operator
     - Special Operator
     - Increment / Decrement Operator

9. **Questions and cross questions in that topic to make them think beyond the topic.**
   1. Explain the Basic Structure of the C Program.
   2. Explain the various Operators available in C Language
   3. Explain the conditional Control Statements in C Language

10. **Final conclusions**

    Students will acquire knowledge basic structure of C Language and Writing C Program

7. Introduction to C

7.1 C – Language History

- C language is a structure oriented programming language, was developed at Bell Laboratories in 1972 by Dennis Ritchie
- C language features were derived from earlier language called “B” (Basic Combined Programming Language – BCPL)
- C language was invented for implementing UNIX operating system
- In 1978, Dennis Ritchie and Brian Kernighan published the first edition “The C Programming Language” and commonly known as K&R C
- In 1983, the American National Standards Institute (ANSI) established a committee to provide a modern, comprehensive definition of C. The resulting definition, the ANSI standard, or “ANSI C”, was completed late 1988.

7.2 C standards

- **C89/C90 standard** – First standardized specification for C language was developed by American National Standards Institute in 1989. C89 and C90 standards refer to the same programming language.
- **C99 standard** – Next revision was published in 1999 that introduced new futures like advanced data types and other changes.

C11 and Embedded C

- C11 standard adds new features to C and library like type generic macros, anonymous structures, improved Unicode support, atomic operations, multi-threading, and bounds-checked functions. It also makes some portions of the existing C99 library optional, and improves compatibility with C++.

- Embedded C includes features not available in normal C like fixed-point arithmetic, named address spaces, and basic I/O hardware addressing
- Operating systems, C compiler and all UNIX application programs are written in C language

- It is also called as procedure oriented programming language
- C language is reliable, simple and easy to use.
- C has been coded in assembly language

7.3 Features of C language:

- Reliability
- Portability
- Flexibility
- Interactivity
- Modularity
- Efficiency and Effectiveness

7.4 Uses of C language:

C language is used for developing system applications that forms major portion of operating systems such as Windows, UNIX and Linux. Below are some examples of C being used.

- Database systems
- Graphics packages
- Word processors
- Spread sheets
- Operating system development
- Compilers and Assemblers
- Network drivers
- Interpreters

Which level the C language is belonging to?

S.no	High Level	Middle Level	Low Level
1	High level languages provides almost everything that the programmer might need to do as already built into the language	Middle level languages don’t provide all the built-in functions found in high level languages, but provides all building blocks that we need to produce the result we want	Low level languages provides nothing other than access to the machines basic instruction set
2	Examples: Java, Python	C, C++	Assembler

C language is a structured language

S.no	Structure oriented	Object oriented	Non structure
1	In this type of language, large	In this type of language, programs	There is no specific

	programs are divided into small programs called functions	are divided into objects	structure for programming this language
2	Prime focus is on functions and procedures that operate on data	Prime focus is on the data that is being operated and not on the functions or procedures	N/A
3	Data moves freely around the systems from one function to another	Data is hidden and cannot be accessed by external functions	N/A
4	Program structure follows “Top Down Approach”	Program structure follows “Bottom UP Approach”	N/A
5	Examples: C, Pascal, ALGOL and Modula-2	C++, JAVA and C# (C sharp)	BASIC, COBOL, FORTRAN

7.5 Key points to remember:

1. C language is structured, middle level programming language developed by Dennis Ritchie
2. Operating system programs such as Windows, Unix, Linux are wriiten by C language
3. C89/C90 and C99 are two standardized editions of C language
4. C has been written in assembly language

Further Reading:

http://www.tutorialspoint.com/cprogramming/

http://www.cprogramming.com

Summary:

- C was invented to write an operating system called UNIX.
- C is a successor of B language which was introduced around 1970
- The language was formalized in 1988 by the American National Standard Institue (ANSI).
- By 1973 UNIX OS almost totally written in C.
- Today C is the most widely used System Programming Language.
- Most of the state of the art software have been implemented using C

Exercises

A.Objective Type Questions:

1. Which of the following language is predecessor to C Programming Language?

○ A ● B ○ BCPL ○ C++

2. C programming language was developed by

○ Dennis Ritchie ○ Ken Thompson ○ Bill Gates ○ Peter Norton

3. C was developed in the year ___

○ 1970 ○ 1972 ○ 1976 ○ 1980

4. C is a ___ language

○ High Level ○ Low Level ○ Middle Level ○ Machine Level

8 Basic Program

We are going to learn a simple “Hello World” C program in this section. Also, all the below topics are explained in this section which are the basics of a C program.

1. C basic program with output and explanation
2. Steps to write C programs and get the output
3. Creation, Compilation and Execution of a C program
    - How to install C compiler and IDE
4. Basic structure of a C program

C Basic Program:

```
#include <stdio.h>

int main()
{
   /* Our first simple C basic program */
   printf("Hello World! ");
   getch();
   return 0;
}
```

Output:

```
Hello World!
```

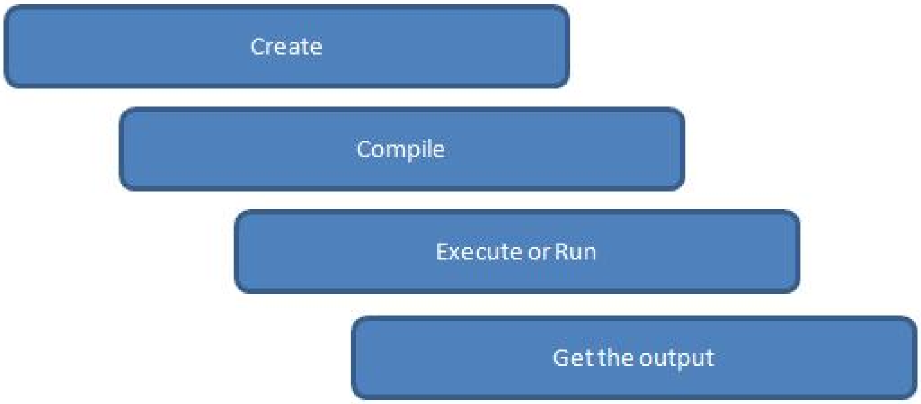
Explanation for above C basic Program:

Let’s see all the sections of the above simple C program line by line.

S.no	Command	Explanation
1	#include <stdio.h>	This is a preprocessor command that includes standard input output header file(stdio.h) from the C library before compiling a C program
2	int main()	This is the main function from where execution of any C program begins.
3	{	This indicates the beginning of the main function.
4	/* some comments */	whatever is given inside the command “/* */” in any C program, won’t be considered for compilation and execution.
5	printf(“Hello World! “);	printf command prints the output onto the screen.
6	getch();	This command waits for any character input from keyboard.
7	return 0;	This command terminates C program (main function) and returns 0.
8	}	This indicates the end of the main function.

8.1 Steps to write C programs and get the output:

Below are the steps to be followed for any C program to create and get the output. This is common to all C program and there is no exception whether its a very small C program or very large C program.



1. **Creation, Compilation and Execution of a C program:**
   Prerequisite:

   - If you want to create, compile and execute C programs by your own, you have to install C compiler in your machine. Then, you can start to execute your own C programs in your machine.
   - You can refer below link for how to install C compiler and compile and execute C programs in your machine.
   - Once C compiler is installed in your machine, you can create, compile and execute C programs as shown in below link.

C – Environment Setup Using IDE tool

C – Environment Setup Using GCC compiler

8.2. Basic structure of C program:

Structure of C program is defined by set of rules called protocol, to be followed by programmer while writing C program. All C programs are having sections/parts which are mentioned below.

1. Documentation section
2. Link Section
3. Definition Section
4. Global declaration section
5. Function prototype declaration section
6. Main function
7. User defined function definition section

Example C program to compare all the sections:

You can compare all the sections of a C program with the below C program.

```
/* C basic structure program   Documentation section
   Author: fresh2refresh.com
   Date    : 01/01/2012
*/
#include <stdio.h>         /* Link section */
int total = 0;   /* Global declaration and definition section */
int sum (int, int);       /* Function declaration section */
int main ()               /* Main function */
{
    printf ("This is a C basic program \n");
    total = sum (1, 1);
    printf ("Sum of two numbers : %d \n", total);
    return 0;
}
int sum (int a, int b)     /* User defined function */
{                        /* definition section    */
    return a + b;
}
```

Output:

```
This is a C basic program
Sum of two numbers : 2
```

8.3 Description for each section of a C program:

- Let us see about each section of a C basic program in detail below.
- Please note that a C program mayn't have all below mentioned sections except main function and link sections.
- Also, a C program structure mayn't be in below mentioned order.

S.No	Sections	Description
1	Documentation section	We can give comments about the program, creation or modified date, author name etc in this section. The characters or words or anything which are given between "/*" and "*/", won't be considered by C compiler for compilation process.These will be ignored by C compiler during compilation. Example : /* comment line1 comment line2 comment 3 */
2	Link Section	Header files that are required to execute a C program are included in this section
3	Definition Section	In this section, variables are defined and values are set to these variables.
4	Global declaration	Global variables are defined in this section. When a variable is to be used

	section	throughout the program, can be defined in this section.
5	Function prototype declaration section	Function prototype gives many information about a function like return type, parameter names used inside the function.
6	Main function	Every C program is started from main function and this function contains two major sections called declaration section and executable section.
7	User defined function section	User can define their own functions in this section which perform particular task as per the user requirement.

C – printf and scanf

- printf() and scanf() functions are inbuilt library functions in C which are available in C library by default. These functions are declared and related macros are defined in “stdio.h” which is a header file.
- We have to include “stdio.h” file as shown in below C program to make use of these printf() and scanf() library functions.

C printf() function:

- printf() function is used to print the “character, string, float, integer, octal and hexadecimal values” onto the output screen.
- We use printf() function with %d format specifier to display the value of an integer variable.
- Similarly %c is used to display character, %f for float variable, %s for string variable, %lf for double and %x for hexadecimal variable.
- To generate a newline,we use “\n” in C printf() statement.

Note:

- C language is case sensitive. For example, printf() and scanf() are different from Printf() and Scanf(). All characters in printf() and scanf() functions must be in lower case.

Example program for C printf() function:

```
#include <stdio.h>

int main()
{
   char  ch = 'A';
   char  str[20] = "fresh2refresh.com";
   float flt = 10.234;
   int no = 150;

   double dbl = 20.123456;
   printf("Character is %c \n", ch);
```

```
    printf("String is %s \n" , str);
    printf("Float value is %f \n", flt);
    printf("Integer value is %d\n" , no);

    printf("Double value is %lf \n", dbl);
    printf("Octal value is %o \n", no);
    printf("Hexadecimal value is %x \n", no);

    return 0;

}
```

Output:

```
Character is A
String is
fresh2refresh.com
Float value is 10.234000
Integer value is 150
Double value is 20.123456
Octal value is 226
Hexadecimal value is 96
```

You can see the output with the same data which are placed within the double quotes of printf statement in the program except

- %d got replaced by value of an integer variable (no),
- %c got replaced by value of a character variable (ch),
- %f got replaced by value of a float variable (flt),
- %lf got replaced by value of a double variable (dbl),
- %s got replaced by value of a string variable (str),
- %o got replaced by a octal value corresponding to integer variable (no),
- %x got replaced by a hexadecimal value corresponding to integer variable
- \n got replaced by a newline.

2. C scanf() function:

- scanf() function is used to read character, string, numeric data from keyboard
- Consider below example program where user enters a character. This value is assigned to the variable "ch" and then displayed.
- Then, user enters a string and this value is assigned to the variable "str" and then displayed.

Example program for printf() and scanf() functions in C:

```
#include <stdio.h>

int main()
{
    char ch;
    char str[100];

    printf("Enter any character \n");
    scanf("%c", &ch);
    printf("Entered character is %c \n", ch);

    printf("Enter any string ( upto 100 character ) \n");
    scanf("%s", &str);
    printf("Entered string is %s \n", str);
}
```

Output:

```
Enter any character
a
Entered character is a
Enter any string ( upto 100 character )
hai
Entered string is hai
```

- The format specifier %d is used in scanf() statement. So that, the value entered is received as an integer and %s for string.
- Ampersand is used before variable name “ch” in scanf() statement as &ch.
- It is just like in a pointer which is used to point to the variable. For more information about how pointer works

Further Reading:

http://www.tutorialspoint.com/cprogramming/
http://www.cprogramming.com

Summary:

- C is a case sensitive programming language. It means in C *printf* and *Printf* will have different meanings.
- C has a free-form line structure. End of each C statement must be marked with a semicolon.
- Multiple statements can be one the same line.
- White Spaces (ie tab space and space bar) are ignored.
- Statements can continue over multiple lines.

Expected Learning Outcome:

Students will acquire knowledge basic structure of C Language.

Exercises

A. Objective Type Questions:

1. C language is available for which of the following Operating Systems?

○ DOS ○ Windows ○ Unix ○ All of these

2. Which of the following symbol is used to denote a pre-processor statement?

○ ! ○ # ○ ~ ○ ;

B.Questions

1.Explain the Basic Structure of the C Program.

9.1 C– Tokens and keywords

C tokens, Identifiers and Keywords are the basics in a C program. All are explained in this page with definition and simple example programs.

9.2 C tokens:

- C tokens are the basic buildings blocks in C language which are constructed together to write a C program.
- Each and every smallest individual units in a C program are known as C tokens.

- C tokens are of six types. They are,

1. Keywords (eg: int, while),
2. Identifiers (eg: main, total),
3. Constants (eg: 10, 20),
4. Strings (eg: “total”, “hello”),
5. Special symbols (eg: (), {}),
6. Operators (eg: +, /,-,*)

C tokens example program:

```
int main()
{
    int x, y, total;
    x = 10, y = 20;
    total = x + y;
    Printf (“Total = %d \n”, total);
}
```

where,

- main – identifier
- {,}, (,) – delimiter
- int – keyword
- x, y, total – identifier
- main, {, }, (,), int, x, y, total – tokens

Do you know how to use C token in real time application programs? We have given simple real time application programs where C token is used. You can refer the below C programs to know how to use C token in real time program.

Real time application C programs for your reference:

1. C program example – Real time Calculator program
2. C program example – Real time Bank Application program

9. 3. Identifiers in C language:

- Each program elements in a C program are given a name called identifiers.
- Names given to identify Variables, functions and arrays are examples for identifiers. eg. x is a name given to integer variable in above program.

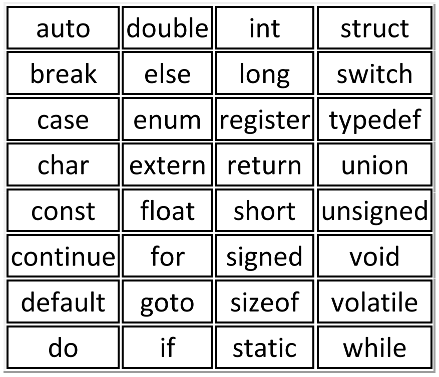
Rules for constructing identifier name in C:

1. First character should be an alphabet or underscore.
2. Succeeding characters might be digits or letter.
3. Punctuation and special characters aren't allowed except underscore.
4. Identifiers should not be keywords.

9.4 Keywords in C language:

- Keywords are pre-defined words in a C compiler.
- Each keyword is meant to perform a specific function in a C program.
- Since keywords are referred names for compiler, they can't be used as variable name.

C language supports 32 keywords which are given below



auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

9.5 C – Constant

- C Constants are also like normal variables. But, only difference is, their values can not be modified by the program once they are defined.
- Constants refer to fixed values. They are also called as literals
- Constants may be belonging to any of the data type.

- Syntax:

const data_type variable_name; (or) const data_type *variable_name;

Types of C constant:

1. Integer constants
2. Real or Floating point constants
3. Octal & Hexadecimal constants
4. Character constants
5. String constants
6. Backslash character constants

S.no	Constant type	data type	Example
1	Integer constants	int unsigned int long int long long int	53, 762, -478 etc 5000u, 1000U etc 483,647 2,147,483,680
2	Real or Floating point constants	float doule	10.456789 600.123456789
3	Octal constant	Int	013 /* starts with 0 */
4	Hexadecimal constant	Int	0×90 /* starts with 0x */
5	character constants	Char	‘A’ , ‘B’, ‘C’
6	string constants	Char	“ABCD” , “Hai”

Rules for constructing C constant:

1. Integer Constants in C:

- An integer constant must have at least one digit.
- It must not have a decimal point.
- It can either be positive or negative.
- No commas or blanks are allowed within an integer constant.
- If no sign precedes an integer constant, it is assumed to be positive.
- The allowable range for integer constants is -32768 to 32767.

2. Real constants in C:

- A real constant must have at least one digit
- It must have a decimal point
- It could be either positive or negative
- If no sign precedes an integer constant, it is assumed to be positive.
- No commas or blanks are allowed within a real constant.

3. Character and string constants in C:

- A character constant is a single alphabet, a single digit or a single special symbol enclosed within single quotes.
- The maximum length of a character constant is 1 character.
- String constants are enclosed within double quotes.

4. Backslash Character Constants in C:

- There are some characters which have special meaning in C language.
- They should be preceded by backslash symbol to make use of special function of them.
- Given below is the list of special characters and their purpose.

Backslash character	Meaning
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\”	Double quote
\’	Single quote
\\	Backslash
\v	Vertical tab
\a	Alert or bell
\?	Question mark
\N	Octal constant (N is an octal constant)
\XN	Hexadecimal constant (N – hex.dcml cnst)

How to use constants in a C program?

- We can define constants in a C program in the following ways.
  1. By “const” keyword
  2. By “#define” preprocessor directive

- Please note that when you try to change constant values after defining in C program, it will through error.

1. Example program using const keyword in C:

```
#include <stdio.h>

void main()
{
   const int  height = 100;               /*int constant*/
   const float number = 3.14;             /*Real constant*/
   const char letter = 'A';               /*char constant*/
   const char letter_sequence[10] = "ABC"; /*string constant*/
   const char backslash_char = '\?';      /*special char cnst*/

   printf("value of height    : %d \n", height );
   printf("value of number : %f \n", number );
   printf("value of letter : %c \n", letter );
   printf("value of letter_sequence : %s \n", letter_sequence);
   printf("value of backslash_char  : %c \n", backslash_char);
}
```

Output:

```
value of height : 100
value of number : 3.140000
value of letter : A
value of letter_sequence : ABC
value of backslash_char : ?
```

2. Example program using #define preprocessor directive in C:

```
C #include <stdio.h>

#define height 100
#define number 3.14
#define letter 'A'
#define letter_sequence "ABC"
#define backslash_char '\?'

void main()
{

   printf("value of height    : %d \n", height );
   printf("value of number : %f \n", number );
   printf("value of letter : %c \n", letter );
   printf("value of letter_sequence : %s \n", letter_sequence);
```

```
    printf("value of backslash_char  : %c \n", backslash_char);

}
```

Output:

```
value of height : 100
value of number : 3.140000
value of letter : A
value of letter_sequence : ABC
value of backslash_char : ?
```

Further Reading:

http://www.tutorialspoint.com/cprogramming/
http://www.cprogramming.com

Summary:

In a passage of text, individual words and punctuation marks are called tokens or lexical units. Similarly, the smallest individual unit in a c program is known as a token or a lexical unit. C tokens can be classified as follows:

1. Keywords
2. Identifiers
3. Constants
4. Strings
5. Special Symbols
6. Operators

Expected Learning Outcome:
Students will acquire knowledge on C-tokens.

Exercises

A.Objective Type Questions:

1. Which of the following are tokens in C?

○ Keywords ○ Variables ○ Constants ○ All of the above

2. Which of the following are tokens in C?

○ Keywords ○ Variables ○ Constants ○ All of the above

B. Questions:

1. Explain about C-tokens.

2. List out keywords in C Language.

10 C – Variable

- C variable is a named location in a memory where a program can manipulate the data. This location is used to hold the value of the variable.
- The value of the C variable may get change in the program.
- C variable might be belonging to any of the data type like int, float, char etc.

10.1 Rules for naming C variable:

1. Variable name must begin with letter or underscore.
2. Variables are case sensitive
3. They can be constructed with digits, letters.
4. No special symbols are allowed other than underscore.
5. sum, height, _value are some examples for variable name

10.2 Declaring & initializing C variable:

- Variables should be declared in the C program before to use.
- Memory space is not allocated for a variable while declaration. It happens only on variable definition.
- Variable initialization means assigning a value to the variable.

S.No	Type	Syntax	Example
1	Variable declaration	data_type variable_name;	int x, y, z; char flat, ch;
2	Variable initialization	data_type variable_name = value;	int x = 50, y = 30; char flag = ‘x’, ch=’l’;

There are three types of variables in C program They are,

1. Local variable
2. Global variable
3. Environment variable

10.3 Example program for local variable in C:

- The scope of local variables will be within the function only.
- These variables are declared within the function and can’t be accessed outside the function.
- In the below example, m and n ivariables are having scope within the main function only. These are not visible to test function.
- Like wise, a and b variables are having scope within the test function only. These are not visible to main function.

```
#include<stdio.h>

void test();

int main()
{
    int m = 22, n = 44;
    // m, n are local variables of main function

    /*m and n variables are having scope
        within this main function only.
        These are not visible to test funtion.*/
    /* If you try to access a and b in this function,
      you will get 'a' undeclared and 'b' undeclared
      error */

    printf("\nvalues : m = %d and n = %d", m, n);

    test();

}

void test()
{
    int a = 50, b = 80;
    // a, b are local variables of test function

    /*a and b variables are having scope
        within this test function only.
        These are not visible to main function.*/
    /* If you try to access m and n in this function,
      you will get 'm' undeclared and 'n' undeclared
      error */

    printf("\nvalues : a = %d and b = %d", a, b);
}
```

Output:

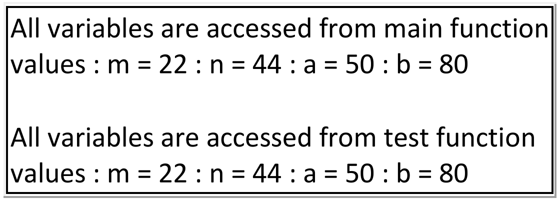
```
values : m = 22 and n = 44
values : a = 50 and b = 80
```

10.4. Example program for global variable in C:

- The scope of global variables will be throughout the program. These variables can be accessed from anywhere in the program.
- This variable is defined outside the main function. So that, this variable is visible to main function and all other sub functions.

```c
#include<stdio.h>

void test();

int m = 22, n = 44;
int a = 50, b = 80;

int main()
{

  printf("All variables are accessed from main function");

  printf("\nvalues : m= %d : n= %d : a= %d : b= %d",m,n,a,b);

  test();

}

void test()
{

  printf("\n\nAll variables are accessed from" \
        " test function");

  printf("\nvalues : m= %d : n= %d : a= %d : b= %d",m,n,a,b);
}
```

Output:



```
All variables are accessed from main function
values : m = 22 : n = 44 : a = 50 : b = 80

All variables are accessed from test function
values : m = 22 : n = 44 : a = 50 : b = 80
```

Difference between variable declaration & definition in C:

S.no	Variable declaration	Variable definition
1	Declaration tells the compiler about data type and size of the variable.	Definition allocates memory for the variable.
2	Variable can be declared many times in a program.	It can happen only one time for a variable in a program.
3	The assignment of properties and identification to a variable.	Assignments of storage space to a variable.

Further Reading:

http://www.tutorialspoint.com/cprogramming/
http://www.cprogramming.com

Summary:

A variable is just a named area of storage that can hold a single value (numeric or character). The C language demands that you declare the name of each variable that you are going to use and its type, or class, before you actually try to do anything with it.

The Programming language C has two main variable types

- Local Variables
- Global Variables

Expected Learning Outcome:
Students will acquire knowledge of C-Variables.

Exercises

A.Objective Type Questions:

1. Which of the following are tokens in C?

○ Keywords ○ Variables ○ Constants ○ All of the above

2. Which of the following are tokens in C?

○ Keywords ○ Variables ○ Constants ○ All of the above

B. Questions:

1. Discuss the following
   a. Local Variable
   b. Global Variable

11 C – Data Types

- C data types are defined as the data storage format that a variable can store a data to perform a specific operation.
- Data types are used to define a variable before to use in a program.
- Size of variable, constant and array are determined by data types.

11.1 C – data types:

There are four data types in C language. They are,

S.no	Types	Data Types
1	Basic data types	int, char, float, double
2	Enumeration data type	Enum
3	Derived data type	pointer, array, structure, union
4	Void data type	Void

11.2 Basic data types in C:

Integer data type:

- Integer data type allows a variable to store numeric values.
- “int” keyword is used to refer integer data type.
- The storage size of int data type is 2 or 4 or 8 byte.
- It varies depend upon the processor in the CPU that we use. If we are using 16 bit processor, 2 byte (16 bit) of memory will be allocated for int data type.
- Like wise, 4 byte (32 bit) of memory for 32 bit processor and 8 byte (64 bit) of memory for 64 bit processor is allocated for int datatype.
- int (2 byte) can store values from -32,768 to +32,767
- int (4 byte) can store values from -2,147,483,648 to +2,147,483,647.
- If you want to use the integer value that crosses the above limit, you can go for “long int” and “long long int” for which the limits are very high.

Note:

- We can’t store decimal values using int data type.
- If we use int data type to store decimal values, decimal values will be truncated and we will get only whole number.
- In this case, float data type can be used to store decimal values in a variable.

11.3 Character data type:

- Character data type allows a variable to store only one character.
- Storage size of character data type is 1. We can store only one character using character data type.
- “char” keyword is used to refer character data type.
- For example, ‘A’ can be stored using char datatype. You can’t store more than one character using char data type.
- Please refer C – Strings topic to know how to store more than one characters in a variable.

11.4 Floating point data type:

Floating point data type consists of 2 types. They are,

1. float
2. double

1. float:

- Float data type allows a variable to store decimal values.
- Storage size of float data type is 4. This also varies depend upon the processor in the CPU as “int” data type.
- We can use up-to 6 digits after decimal using float data type.
- For example, 10.456789 can be stored in a variable using float data type.

2. double:

- Double data type is also same as float data type which allows up-to 10 digits after decimal.
- The range for double datatype is from 1E–37 to 1E+37.

sizeof() function in C:

sizeof() function is used to find the memory space allocated for each C data types.

```
#include <stdio.h>
#include <limits.h>

int main()
{

  int a;
  char b;
  float c;
  double d;
  printf("Storage size for int data type:%d \n",sizeof(a));
  printf("Storage size for char data type:%d \n",sizeof(b));
  printf("Storage size for float data type:%d \n",sizeof(c));
  printf("Storage size for double data type:%d\n",sizeof(d));
  return 0;
}
```

Output:

```
Storage size for int data type:4
Storage size for char data type:1
Storage size for float data type:4
Storage size for double data type:8
```

11.5 Modifiers in C:

- The amount of memory space to be allocated for a variable is derived by modifiers.
- Modifiers are prefixed with basic data types to modify (either increase or decrease) the amount of storage space allocated to a variable.
- For example, storage space for int data type is 4 byte for 32 bit processor. We can increase the range by using long int which is 8 byte. We can decrease the range by using short int which is 2 byte.

- There are 5 modifiers available in C language. They are,

1. short
2. long
3. signed
4. unsigned
5. long long

- Below table gives the detail about the storage size of each C basic data type in 16 bit processor. Please keep in mind that storage size and range for int and float datatype will vary depend on the CPU processor (8,16, 32 and 64 bit)

S.No	C Data types	storage Size	Range
1	Char	1	–127 to 127
2	Int	2	–32,767 to 32,767
3	Float	4	1E–37 to 1E+37 with six digits of precision
4	Double	8	1E–37 to 1E+37 with ten digits of precision
5	long double	10	1E–37 to 1E+37 with ten digits of precision
6	long int	4	–2,147,483,647 to 2,147,483,647
7	short int	2	–32,767 to 32,767
8	unsigned short int	2	0 to 65,535
9	signed short int	2	–32,767 to 32,767
10	long long int	8	–(2power(63) –1) to 2(power)63 –1
11	signed long int	4	–2,147,483,647 to 2,147,483,647
12	unsigned long int	4	0 to 4,294,967,295
13	unsigned long long int	8	2(power)64 –1

11.6 Enumeration data type in C:

- Enumeration data type consists of named integer constants as a list.
- It start with 0 (zero) by default and value is incremented by 1 for the sequential identifiers in the list.

Enum syntax in C:

enum identifier [optional{ enumerator-list }];

- Enum example in C:

enum month { Jan, Feb, Mar }; or
/* Jan, Feb and Mar variables will be assigned to 0, 1 and 2 respectively by default */
enum month { Jan = 1, Feb, Mar };
/* Feb and Mar variables will be assigned to 2 and 3 respectively by default */
enum month { Jan = 20, Feb, Mar };
/* Jan is assigned to 20. Feb and Mar variables will be assigned to 21 and 22 respectively by default */

- The above enum functionality can also be implemented by "#define" preprocessor directive as given below. Above enum example is same as given below.

```
#define Jan 20;
#define Feb 21;
#define Mar 22;
```

11.7 C – enum example program:

```
#include <stdio.h>

int main()
{
  enum MONTH { Jan = 0, Feb, Mar };

  enum MONTH month = Mar;

  if(month == 0)
    printf("Value of Jan");
  else if(month == 1)
    printf("Month is Feb");
  if(month == 2)
    printf("Month is Mar");}
```

Output:

```
Month is March
```

11.8 Derived data type in C:

- Array, pointer, structure and union are called derived data type in C language.

11.9 Void data type in C:

- Void is an empty data type that has no value.
- This can be used in functions and pointers.

Further Reading:

http://www.tutorialspoint.com/cprogramming/
http://www.cprogramming.com

Summary:

C has a concept of 'data types' which are used to define a variable before its use. The definition of a variable will assign storage for the variable and define the type of data that will be held in the location.

The value of a variable can be changed any time. C has the following basic built-in datatypes.

- int
- float
- double
- char
- void

Expected Learning Outcome:

Students will acquire knowledge of C-datatypes.

Exercises

A.Objective Type Questions:

1. What will be the maximum size of a float variable?

○ 1 byte ○ 2 bytes ○ 4 bytes ○ 8 bytes

2. What will be the maximum size of a double variable?

○ 1 byte ○ 4 bytes ○ 8 bytes ○ 16 bytes

B. Questions:

1. Explain in detail about C-Datatypes.

2. Discuss the following

   a. Derived Datatype

   b. User defined Datatype

12 C – Operators and Expressions

- The symbols which are used to perform logical and mathematical operations in a C program are called C operators.
- These C operators join individual constants and variables to form expressions.
- Operators, functions, constants and variables are combined together to form expressions.
- Consider the expression A + B * 5. where, +, * are operators, A, B are variables, 5 is constant and A + B * 5 is an expression.

12.1 Types of C operators:

C language offers many types of operators. They are,

1. Arithmetic operators
2. Assignment operators
3. Relational operators
4. Logical operators
5. Bit wise operators
6. Conditional operators (ternary operators)
7. Increment/decrement operators
8. Special operators

S.no	Types of Operators	Description
1	Arithmetic_operators	These are used to perform mathematical calculations like addition, subtraction, multiplication, division and modulus
2	Assignment_operators	These are used to assign the values for the variables in C programs.
3	Relational operators	These operators are used to compare the value of two variables.
4	Logical operators	These operators are used to perform logical operations on the given two variables.
5	Bit wise operators	These operators are used to perform bit operations on given two variables.
6	Conditional (ternary) operators	Conditional operators return one value if condition is true and returns another value is condition is false.
7	Increment/decrement operators	These operators are used to either increase or decrease the value of the variable by one.
8	Special operators	&, *, sizeof( ) and ternary operators.

12.2 C- Arithmetic Operators

Arithmetic Operators in C:

- C Arithmetic operators are used to perform mathematical calculations like addition, subtraction, multiplication, division and modulus in C programs.

S.no	Arithmetic Operators	Operation	Example
1	+	Addition	A+B
2	-	Subtraction	A-B
3	*	multiplication	A*B
4	/	Division	A/B
5	%	Modulus	A%B

Example program for C arithmetic operators:

- In this example program, two values “40” and “20” are used to perform arithmetic operations such as addition, subtraction, multiplication, division, modulus and output is displayed for each operation.

```
#include <stdio.h>
int main()
{
   int a=40,b=20, add,sub,mul,div,mod;

   add = a+b;
   sub = a-b;
   mul = a*b;
   div = a/b;
   mod = a%b;

   printf("Addition of a, b is   : %d\n", add);
   printf("Subtraction of a, b is   : %d\n", sub);
   printf("Multiplication of a, b is   : %d\n", mul);
   printf("Division of a, b is   : %d\n", div);
   printf("Modulus of a, b is   : %d\n", mod);
}
```

Output:

```
Addition of a, b is : 60
Subtraction of a, b is : 20
Multiplication of a, b is : 800
Division of a, b is : 2
Modulus of a, b is : 0
```

12.3 C – Assignment Operators

Assignment operators in C:

- In C programs, values for the variables are assigned using assignment operators.
- For example, if the value “10” is to be assigned for the variable “sum”, it can be assigned as “sum = 10;”
- Other assignment operators in C language are given below.

Operators		Example	Explanation
Simple assignment operator	=	sum=10	10 is assigned to variable sum
Compound assignment operators	+=	sum+=10	This is same as sum=sum+10
	-=	sum-=10	This is same as sum = sum-10
	*=	sum*=10	This is same as sum = sum*10
	/+	sum/=10	This is same as sum = sum/10
	%=	sum%=10	This is same as sum = sum%10
	&=	sum&=10	This is same as sum = sum&10
	^=	sum^=10	This is same as sum = sum^10

Example program for C assignment operators:

- In this program, values from 0 – 9 are summed up and total “45” is displayed as output.
- Assignment operators such as “=” and “+=” are used in this program to assign the values and to sum up the values.

```
# include <stdio.h>
int main()
{
   int Total=0,i;
   for(i=0;i<10;i++)
   {
      Total+=i; // This is same as Total = Toatal+i
   }
   printf("Total = %d", Total);
}
```

Output:

```
Total = 45
```

12.4 C – Relational Operators

Relational operators in C:

- Relational operators are used to find the relation between two variables. i.e. to compare the values of two variables in a C program.

S.no	Operators	Example	Description
1	>	x > y	x is greater than y
2	<	x < y	x is less than y
3	>=	x >= y	x is greater than or equal to y
4	<=	x <= y	x is less than or equal to y
5	==	x == y	x is equal to y
6	!=	x != y	x is not equal to y

Example program for relational operators in C:

- In this program, relational operator (==) is used to compare 2 values whether they are equal are not.
- If both values are equal, output is displayed as ” values are equal”. Else, output is displayed as “values are not equal”.
- Note : double equal sign (==) should be used to compare 2 values. We should not single equal sign (=).

```
#include <stdio.h>
int main()
{
  int m=40,n=20;
  if (m == n)
  {
    printf("m and n are equal");
  }
  else
  {
    printf("m and n are not equal");
  }
}
```

Output:

```
m and n are not equal
```

12.5 C – Logical Operators

Logical operators in C:

- These operators are used to perform logical operations on the given expressions.
- There are 3 logical operators in C language. They are, logical AND (&&), logical OR (||) and logical NOT (!).

S.no	Operators	Name	Example	Description
1	&&	logical AND	(x>5)&&(y<5)	It returns true when both conditions are true
2	||	logical OR	(x>=10)||(y>=10)	It returns true when at-least one of the condition is true
3	!	logical NOT	!((x>5)&&(y<5))	It reverses the state of the operand "((x>5) && (y<5))" If "((x>5) && (y<5))" is true, logical NOT operator makes it false

Example program for logical operators in C:

```
#include <stdio.h>
int main()
{
    int m=40,n=20;
    int o=20,p=30;

    if (m>n && m !=0)
    {
        printf("&& Operator : Both conditions are true\n");
    }
    if (o>p || p!=20)
    {
        printf("|| Operator : Only one condition is true\n");
    }
    if (!(m>n && m !=0))
    {
       printf("! Operator  : Both conditions are true\n");
    }
    else
    {
       printf("! Operator  : Both conditions are true. " \
             "But, status is inverted as   false\n");
    }
}
```

Output:

```
&& Operator : Both conditions are true
|| Operator : Only one condition is true
! Operator : Both conditions are true. But, status is inverted as false
```

- In this program, operators (&&, || and !) are used to perform logical operations on the given expressions.
- **&& operator** – "if clause" becomes true only when both conditions (m>n and m! =0) is true. Else, it becomes false.
- **|| Operator** – "if clause" becomes true when any one of the condition (o>p || p!=20) is true. It becomes false when none of the condition is true.
- **! Operator** – It is used to reverses the state of the operand.
- If the conditions (m>n && m!=0) is true, true (1) is returned. This value is inverted by "!" operator.
- So, "! (m>n and m! =0)" returns false (0).

12.6 C – Bit wise Operators

Bit wise operators in C:

- These operators are used to perform bit operations. Decimal values are converted into binary values which are the sequence of bits and bit wise operators work on these bits.
- Bit wise operators in C language are & (bitwise AND), | (bitwise OR), ~ (bitwise OR), ^ (XOR), << (left shift) and >> (right shift).

Truth table for bit wise operation

X	y	x|y	x & y	x ^ y
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

Bit wise operators

Operator symbol	Operator name
&	Bitwise AND
|	Bitwise OR
~	Bitwise NOT
^	XOR
<<	Left Shift
>>	Right Shift

- Consider x=40 and y=80. Binary form of these values are given below.

x = 00101000
y= 01010000

- All bit wise operations for x and y are given below.

x&y = 00000000 (binary) = 0 (decimal)
x|y = 01111000 (binary) = 120 (decimal)
~x =1111111111111111111111111111111111111111111111111111111111010111
= -41 (decimal)
x^y = 01111000 (binary) = 120 (decimal)
x << 1 = 01010000 (binary) = 80 (decimal)
x >> 1 = 00010100 (binary) = 20 (decimal)

Note:

- **Bit wise NOT :** Value of 40 in binary is 0000000000000000000000000000000000000000000000000000000000101000. So, all 0's are converted into 1's in bit wise NOT operation.
- **Bit wise left shift and right shift :** In left shift operation "x << 1 ", 1 means that the bits will be left shifted by one place. If we use it as "x << 2 ", then, it means that the bits will be left shifted by 2 places.

Example program for bit wise operators in C:

- In this example program, bit wise operations are performed as shown above and output is displayed in decimal format.

```
#include <stdio.h>
int main()
{
   int m=40,n=80,AND_opr,OR_opr,XOR_opr,NOT_opr ;

   AND_opr = (m&n);
   OR_opr = (m|n);
   NOT_opr = (~m);
   XOR_opr = (m^n);

   printf("AND_opr value = %d\n",AND_opr );
   printf("OR_opr value = %d\n",OR_opr );
   printf("NOT_opr value = %d\n",NOT_opr );
   printf("XOR_opr value = %d\n",XOR_opr );
   printf("left_shift value = %d\n", m << 1);
   printf("right_shift value = %d\n", m >> 1);
}
```

Output:

```
AND_opr value = 0
OR_opr value = 120
NOT_opr value = -41
XOR_opr value = 120
left_shift value = 80
right_shift value = 20
```

12.7 C – Conditional Operators

Conditional or ternary operators in C:

- Conditional operators return one value if condition is true and returns another value is condition is false.
- This operator is also called as ternary operator.

Syntax : (Condition? true_value: false_value);
Example : (A > 100 ? 0 : 1);

- In above example, if A is greater than 100, 0 is returned else 1 is returned. This is equal to if else conditional statements.

Example program for conditional/ternary operators in C:

```
#include <stdio.h>
int main()
{
       int x=1, y ;
       y = ( x ==1 ? 2 : 0 ) ;
       printf("x value is %d\n", x);
       printf("y value is %d", y);
}
```

Output:

```
x value is 1
y value is 2
```

12.8 C – Increment/decrement Operators

- Increment operators are used to increase the value of the variable by one and decrement operators are used to decrease the value of the variable by one in C programs.

- Syntax:

Increment operator :
++var_name;(or) var_name++;
Decrement operator :
– - var_name; (or) var_name – -;

- Example:

Increment operator :
++i;i++;
Decrement operator: – - i ; i – - ;

Example program for increment operators in C:

- In this program, value of “i” is incremented one by one from 1 up to 9 using “i++” operator and output is displayed as “1 2 3 4 5 6 7 8 9”.

```
//Example for increment operators

#include <stdio.h>
int main()
{
    int i=1;
    while(i<10)
    {
       printf("%d ",i);
       i++;
    }
}
```

Output:

```
1 2 3 4 5 6 7 8 9
```

Example program for decrement operators in C:

- In this program, value of “I” is decremented one by one from 20 up to 11 using “i–” operator and output is displayed as “20 19 18 17 16 15 14 13 12 11”.

```
//Example for decrement operators

#include <stdio.h>
int main()
{
   int i=20;
   while(i>10)
   {
       printf("%d ",i);
       i--;
   }
}
```

Output:

```
20 19 18 17 16 15 14 13 12 11
```

Difference between pre/post increment & decrement operators in C:

- Below table will explain the difference between pre/post increment and decrement operators in C.

S.no	Operator type	Operator	Description
1	Pre increment	++i	Value of i is incremented before assigning it to variable i.
2	Post increment	i++	Value of i is incremented after assigning it to variable i.
3	Pre decrement	– –i	Value of i is decremented before assigning it to variable i.
4	Post decrement	i– –	Value of i is decremented after assigning it to variable i.

Example program for pre – increment operators in C:

```
//Example for increment operators

#include <stdio.h>
int main()
{
          int i=0;
          while(++i < 5 )
          {
              printf("%d ",i);
          }
          return 0;
}
```

Output:

```
1 2 3 4
```

- Step 1 : In above program, value of “i” is incremented from 0 to 1 using pre-increment operator.
- Step 2 : This incremented value “1” is compared with 5 in while expression.
- Step 3 : Then, this incremented value “1” is assigned to the variable “i”.
- Above 3 steps are continued until while expression becomes false and output is displayed as “1 2 3 4”.

Example program for post – increment operators in C:

```
#include <stdio.h>
int main()
{
        int i=0;
        while(i++ < 5 )
        {
           printf("%d ",i);
        }
        return 0;
}
```

Output:

```
1 2 3 4 5
```

- Step 1 : In this program, value of i ”0” is compared with 5 in while expression.
- Step 2 : Then, value of “i” is incremented from 0 to 1 using post-increment operator.
- Step 3 : Then, this incremented value “1” is assigned to the variable “i”.
- Above 3 steps are continued until while expression becomes false and output is displayed as “1 2 3 4 5”.

Example program for pre - decrement operators in C:

```
#include <stdio.h>
int main()
{
        int i=10;
        while(--i > 5 )
        {
           printf("%d ",i);
        }
        return 0;
}
```

Output:

```
9 8 7 6
```

- Step 1 : In above program, value of “i” is decremented from 10 to 9 using pre-decrement operator.
- Step 2 : This decremented value “9” is compared with 5 in while expression.
- Step 3 : Then, this decremented value “9” is assigned to the variable “i”.
- Above 3 steps are continued until while expression becomes false and output is displayed as “9 8 7 6”.

Example program for post - decrement operators in C:

```
#include <stdio.h>
int main()
{
        int i=10;
        while(i-- > 5 )
        {
            printf("%d ",i);
        }
        return 0;
}
```

Output:

```
9 8 7 6 5
```

- Step 1 : In this program, value of i ”10” is compared with 5 in while expression.
- Step 2 : Then, value of “i” is decremented from 10 to 9 using post-decrement operator.
- Step 3 : Then, this decremented value “9” is assigned to the variable “i”.
- Above 3 steps are continued until while expression becomes false and output is displayed as “9 8 7 6 5”.

12.9 C – Special Operators

Special Operators in C:

- Below are some of special operators that C language offers.

S.no	Operators	Description
1	&	This is used to get the address of the variable. Example : &a will give address of a.
2	*	This is used as pointer to a variable. Example : * a where, * is pointer to the variable a.
3	Sizeof ()	This gives the size of the variable. Example : size of (char) will give us 1.

Example program for & and * operators in C:

- In this program, “&” symbol is used to get the address of the variable and “*” symbol is used to get the value of the variable that the pointer is pointing to. Please refer C – pointer topic to know more about pointers.

```
#include <stdio.h>

int main()
{
        int *ptr, q;
        q = 50;
        /* address of q is assigned to ptr       */
        ptr = &q;
        /* display q's value using ptr variable */
        printf("%d", *ptr);
        return 0;
}
```

Output:

```
50
```

Example program for sizeof() operator in C:

- sizeof() operator is used to find the memory space allocated for each C data types.

```
#include <stdio.h>
#include <limits.h>

int main()
{

   int a;
   char b;
   float c;
   double d;
   printf("Storage size for int data type:%d \n",sizeof(a));
   printf("Storage size for char data type:%d \n",sizeof(b));
   printf("Storage size for float data type:%d \n",sizeof(c));
   printf("Storage size for double data type:%d\n",sizeof(d));
   return 0;
}
```

Output:

```
Storage size for int data type:4
Storage size for char data type:1
Storage size for float data type:4
Storage size for double data type:8
```

Further Reading:

http://www.tutorialspoint.com/cprogramming/
http://www.cprogramming.com

Summary:

C language supports following type of operators.

- Arithmetic Operators
- Logical (or Relational) Operators
- Bitwise Operators
- Assignment Operators
- Misc Operators

Expected Learning Outcome:
Students will acquire knowledge on C Operators and Expressions.

Exercises
A.Objective Type Questions:

1. Which of the following is an example of compounded assignment statement?

○ a = 5 ○ a += 5 ○ a = b = c ○ a = b

2. The operator && is an example for ___ operator.

○ Assignment ○ Increment ○ Logical ○ Rational

3. The operator & is used for

○ Bitwise AND ○ Bitwise OR ○ Logical AND ○ Logical OR

4. The operator / can be applied to

○ integer values ○ float values ○ double values ○ All of these

5. The equality operator is represented by

○ := ○ .EQ. ○ = ○ ==

B. Questions:

1. Explain in detail about C-Operators with examples.
2. Discuss the following
 a. Conditional Operators
 b. Increment and Decrement Operators

13 C – Decision Control statement

- In decision control statements (C if else and nested if), group of statements are executed when condition is true. If condition is false, then else part statements are executed.
- There are 3 types of decision making control statements in C language. They are,

1. if statements
2. if else statements
3. nested if statements

13 .1 If", "else" and "nested if" decision control statements in C:

- Syntax for each C decision control statements are given in below table with description.

Decision control statements	Syntax	Description

if	if (condition)
	{ Statements; }
		In these type of statements, if condition is true, then respective block of code is executed.

if…else	if (condition)
	{ Statement1; Statement2;}
	else
	{ Statement3; Statement4; }
		In these type of statements, group of statements are executed when condition is true. If condition is false, then else part statements are executed.

nested if	if (condition1){ Statement1; }
	else if (condition2)
	{ Statement2; }
	else Statement 3;
		If condition 1 is false, then condition 2 is checked and statements are executed if it is true. If condition 2 also gets failure, then else part is executed.

Example program for if statement in C:

In "if" control statement, respective block of code is executed when condition is true.

```
int main()
{
  int m=40,n=40;
  if (m == n)
  {
     printf("m and n are equal");
  }
}
```

Output:

```
m and n are equal
```

Example program for if else statement in C:

In C if else control statement, group of statements are executed when condition is true. If condition is false, then else part statements are executed.

```
#include <stdio.h>
int main()
{
  int m=40,n=20;
  if (m == n) {
      printf("m and n are equal");
  }
  else {
        printf("m and n are not equal");
  }

}
```

Output:

```
m and n are not equal
```

Example program for nested if statement in C:

- In “nested if” control statement, if condition 1 is false, then condition 2 is checked and statements are executed if it is true.
- If condition 2 also gets failure, then else part is executed.

```
#include <stdio.h>
int main()
{
  int m=40,n=20;
  if (m>n) {
      printf("m is greater than n");
  }
  else if(m<n) {
        printf("m is less than n");
  }
  else {
        printf("m is equal to n");
  }
}
```

Output:

```
m is greater than n
```

13.2 C – Loop control statements

Loop control statements in C are used to perform looping operations until the given condition is true. Control comes out of the loop statements once condition becomes false.

Types of loop control statements in C:

There are 3 types of loop control statements in C language. They are,

1. for
2. while
3. do-while

- Syntax for each C loop control statements are given in below table with description.

S.no	Loop Name	Syntax	Description

1	for	for (exp1; exp2; expr3)
		{ statements; }
			Where,
			exp1 – variable initialization
			(Example: i=0, j=2, k=3)
			exp2 – condition checking
			(Example: i>5, j<3, k=3)
			exp3 – increment/decrement
			(Example: ++i, j–, ++k)

2	while	while (condition)
		{ statements; }
			where,
			condition might be a>5, i<10

3	do while	do { statements; }
		while (condition);
			where,
			condition might be a>5, i<10

Example program (for loop) in C:

In for loop control statement, loop is executed until condition becomes false.

```
#include <stdio.h>

int main()
{
  int i;

  for(i=0;i<10;i++)
  {
      printf("%d ",i);  }    }
```

Output:

```
0 1 2 3 4 5 6 7 8 9
```

Example program (while loop) in C:

In while loop control statement, loop is executed until condition becomes false.

```
#include <stdio.h>

int main()
{
  int i=3;

  while(i<10)
  {
     printf("%d\n",i);
     i++;
  }

}
```

Output:

```
3 4 5 6 7 8 9
```

Example program (do while loop) in C:

In do..while loop control statement, while loop is executed irrespective of the condition for first time. Then 2nd time onwards, loop is executed until condition becomes false.

```
#include <stdio.h>

int main()
{
  int i=1;

  do
  {
     printf("Value of i is %d\n",i);
     i++;
  }while(i<=4 && i>=2);

}
```

Output:

```
Value of i is 1
Value of i is 2
Value of i is 3
Value of i is 4
```

Difference between while & do while loops in C:

S.no	while	do while
1	Loop is executed only when condition is true.	Loop is executed for first time irrespective of the condition. After executing while loop for first time, then condition is checked.

13.3 C – Case control statements

The statements which are used to execute only specific block of statements in a series of blocks are called case control statements.

There are 4 types of case control statements in C language. They are,

1. switch
2. break
3. continue
4. goto

switch case statement in C:

- Switch case statements are used to execute only specific case statements based on the switch expression.
- Below is the syntax for switch case statement.

```
switch (expression)
{
        case label1:   statements;
                       break;
        case label2:   statements;
                       break;
        default:       statements;
                       break;
}
```

Example program for switch..case statement in C:

```c
#include <stdio.h>

int main ()
{
  int value = 3;

  switch(value)
  {
  case 1:
    printf("Value is 1 \n" );
    break;
  case 2:
    printf("Value is 2 \n" );
    break;
  case 3:
    printf("Value is 3 \n" );
    break;
  case 4:
    printf("Value is 4 \n" );
    break;
  default :
    printf("Value is other than 1,2,3,4 \n" );
  }

  return 0;
}
```

Output:

```
Value is 3
```

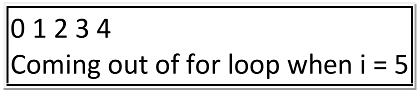
break statement in C:

- Break statement is used to terminate the while loops, switch case loops and for loops from the subsequent execution.
- Syntax: break;

Example program for break statement in C:

```
#include <stdio.h>

int main()
{
  int i;

  for(i=0;i<10;i++)
  {
      if(i==5)
      {
          printf("\nComing out of for loop when i = 5");
    break;
      }
      printf("%d ",i);
  }

}
```

Output:



```
0 1 2 3 4
Coming out of for loop when i = 5
```

Continue statement in C:

- Continue statement is used to continue the next iteration of for loop, while loop and do-while loops. So, the remaining statements are skipped within the loop for that particular iteration.
- Syntax : continue;

Example program for continue statement in C:

```
#include <stdio.h>

int main()
{
  int i;

  for(i=0;i<10;i++)
  {
    if(i==5 || i==6)
    {
       printf("\nSkipping %d from display using " \
             "continue statement \n",i);
  continue;
    }
    printf("%d ",i);
 }

}
```

Output:

```
0 1 2 3 4
Skipping 5 from display using continue statement
Skipping 6 from display using continue statement
7 8 9
```

goto statement in C:

- goto statements is used to transfer the normal flow of a program to the specified label in the program.
- Below is the syntax for goto statement in C.

```
{
        …….
        go to label;
        …….
        …….
        LABEL:
        statements;
}
```

Example program for goto statement in C:

```
#include <stdio.h>

int main()
{
  int i;

  for(i=0;i<10;i++)
  {
    if(i==5)
    {
        printf("\nWe are using goto statement when i = 5");
        goto HAI;
     }
     printf("%d ",i);
  }

HAI : printf("\nNow, we are inside label name \"hai\" \n");

}
```

Output:

```
0 1 2 3 4
We are using goto statement when i = 5
Now, we are inside label name “hai”
```

Further Reading: List the chapters in Reference books/ web resources

http://www.tutorialspoint.com/cprogramming/
http://www.cprogramming.com

Summary:

C provides two styles of flow control:

- Branching
- Looping

Branching is deciding what actions to take and looping is deciding how many times to take a certain action.

Expected Learning Outcome:

Students will acquire knowledge on Decision control statements and loop control statements.

Exercises

A.Objective Type Questions:

1.What is the output of the following program?

```
#include<stdio.h>
void main()
{
   char c= 'a';
   switch(c)
   {
      case 65 : printf("inside case A,");
              break;
      case 97 : printf("inside case a,");
              break;
      default: printf("inside default case");
             break;
   }}
```

A. inside default case

B. inside case a,

C. Compilation Error

D. inside case A,

2.What is the output of the following program?

```
#include<stdio.h>
void main()
{
   int i=10;
   switch(i)
   {
      case 1: printf(" i=1");
              break;
      case 10: printf(" i=10");
      case 11: printf(" i=11");
              break;
      case 12: printf(" i=12");
    }
}
```

A. i=10 i=11 i=12

B. i=1 i=10 i=11 i=12

C. i=10 i=11

D. None of the Above

COURSE PLAN

Select a course (a theory subject): **Programming for Problem Solving**

Select a unit: **Module III**

Select a Topic: **Arrays and Functions**

1. **Objectives**

   To study about the concepts of Arrays, types of arrays

2. **Outcomes**

   Students will acquire knowledge on array declaration and (runtime & Compilation) initialization.

3. **Pre-requisites**

   Know the concept of basic data types in C Language

4. **Terminology used other than normal known scientific / engg terms and their fundamental explanations / relations**

Plan for the lecture delivery

1. **How to plan for delivery – black board / ppt / animated ppt / (decide which is good for this topic)**

   Power Point Presentation

2. **How to explain the definition and terms with in it**

   Explain the memory allocation using Arrays, types of arrays and its initialization concepts

3. **How to start the need for any derivation / procedure / experiment / case study**

   Not Applicable

4. **Physical meaning of math equations / calculations**

   Not Applicable

5. **Units and their physical meaning to make them understand practical reality and comparison between different units**

   Understand the array concept easily apply the array concept in Array of structure in the next unit.

6. **What is the final conclusion?**

Students will acquire knowledge on array declaration and (runtime & Compilation) initialization.

7. **How to put it in a nut shell**

One Dimension Array, Two Dimension Array, Multi Dimension Array, User define function, Standard Library Function.

8. **Important points for understanding / memorizing / make it long lasting**

- Array is a Group of Similar Data Items. Each Items accessed using the Array index number
- Three Types of Array one, Two, Multi Dimension Array
- Function is a group of statement to solve a specific task
- Two Types of function 1. User Define function 2. Standard Library Function.
- Function call the same function itself is called recursive function.

9. **Questions and cross questions in that topic to make them think beyond the topic.**

A 1. Array is a container of similar ……………………………

a) symbols b) data elements c) variables d) constants

B 1. Define an array

2. What are all the types of array available?

3. Give the syntax of array declaration.

C 1. Explain briefly about arrays.

D

1. Define a function?
2. Define formal parameter
3. Define Actual Parameter

1. Explain briefly about functions in C programming
2. Compare function call by reference with call by value.

10. **Final conclusions**

Students will acquire knowledge on array declaration and (runtime & Compilation) initialization.

14.1 Introduction.

The C Programming language generally uses the variables which are nothing but 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 limitations are it can’t store more than one value at a time. The limitation of variables could be avoided by using the arrays.

Arrays.

C programming language provides a data structure called the array, which can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

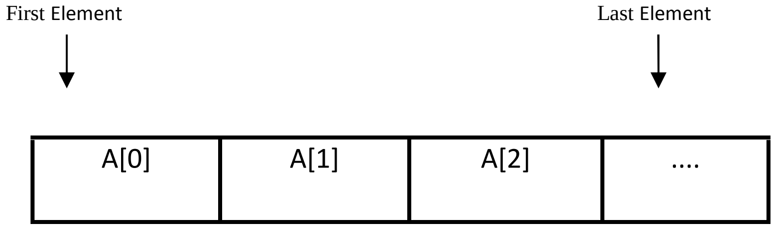


Fig 3.1 Arrays

14.2. Array Declaration.

To declare an array in C, a programmer specifies the type of the elements and the number of elements required by an array as follows,

Syntax

<type> <array Name> [array Size];

This is called a single-dimensional array. The **arraySize** must be an integer constant greater than zero and type can be any valid C data type. For example, to declare a 10-element array called balance of type double, use this statement:

double balance[10];

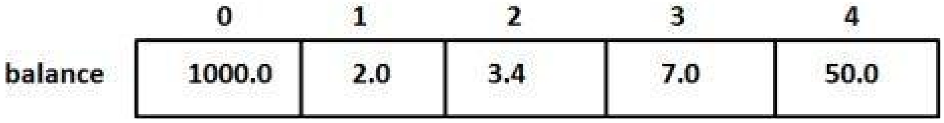
14.3. Initializing Arrays

You can initialize array in C either one by one or using a single statement as follows:

double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};

The number of values between braces { } cannot be larger than the number of elements that we declare for the array between square brackets [ ].

Following is the pictorial representation of the same array we discussed above:



	0	1	2	3	4
balance	1000.0	2.0	3.4	7.0	50.0

Fig 3.2 Array with elements

14.4. Accessing Array Elements

An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example:

double salary = balance[9];

The above statement will take 10th element from the array and assign the value to salary variable. Following is an example which will use all the above mentioned three concepts viz. declaration, assignment and accessing arrays:

```
  #include <stdio.h>
  int main ()
  {
  int n[ 10 ];                      /* n is an array of 10 integers */
  int i,j;
  /* initialize elements of array n to 0 */
  for ( i = 0; i < 10; i++ )
  {
  n[ i ] = i + 100;                 /* set element at location i to i + 100 */
  }
  /* output each array element's value */
  for (j = 0; j < 10; j++ )
  {
  printf("Element[%d] = %d\n", j, n[j] );
  }
  return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
Element[0] = 100
Element[1] = 101
Element[2] = 102
Element[3] = 103
Element[4] = 104
Element[5] = 105
Element[6] = 106
Element[7] = 107
Element[8] = 108
Element[9] = 109
```

14.5 Types of Arrays.

Arrays are classified with their allocation and usage, namely

- Two dimensional Array
- Multi- dimensional Array

14.5.1 Multi-dimensional Arrays

C programming language allows **multidimensional arrays**. Here is the general form of a multidimensional array declaration:

```
type name[size1][size2].  .  .[sizeN];
```

For example, the following declaration creates a three dimensional 5 . 10 . 4 integer array:

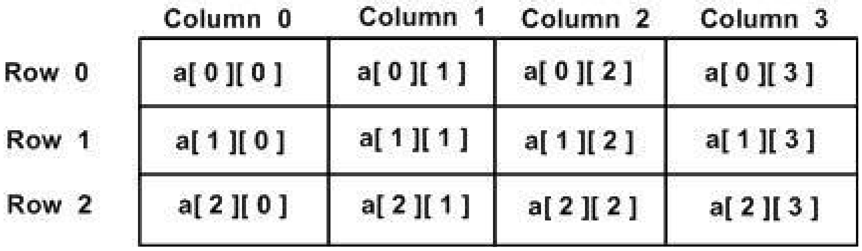
```
int threedim[5][10][4];
```

14.5.2 Two-Dimensional Arrays

The simplest form of the **multidimensional** array is the **two-dimensional** array. A two-dimensional array is, in essence, a list of one-dimensional arrays. To declare a two-dimensional integer array of size x, y you would write something as follows:

type arrayName [x][y];

Where type can be any valid C data type and **arrayName** will be a valid C identifier. A two-dimensional array can be think as a table which will have x number of rows and y number of columns. A 2-dimentional array **a**, which contains three rows and four columns can be shown as below:



	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Thus, every element in array a is identified by an element name of the form a[i][j], where a is the name of the array, and i and j are the subscripts that uniquely identify each element in '**a**'.

Initializing Two-Dimensional Arrays

Multidimensional arrays may be initialized by specifying bracketed values for each row. Following is an array with 3 rows and each row has 4 columns.

```
int a[3][4] = {
{0, 1, 2, 3} ,          /* initializers for row indexed by 0 */
{4, 5, 6, 7} ,          /* initializers for row indexed by 1 */
{8, 9, 10, 11}         /* initializers for row indexed by 2 */
};
```

The nested braces, which indicate the intended row, are optional. The following initialization is equivalent to previous example:

int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};

Accessing Two-Dimensional Array Elements

An element in 2-dimensional array is accessed by using the subscripts, i.e., row index and column index of the array. For example:

int val = a[2][3];

The above statement will take 4th element from the 3rd row of the array. You can verify it in the above diagram. Let us check below program where we have used nested loop to handle a two dimensional array:

```
#include <stdio.h>
int main ()
{
/* an array with 5 rows and 2 columns*/
int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};
int i, j;
/* output each array element's value */
for ( i = 0; i < 5; i++ )
{
for ( j = 0; j < 2; j++ )
{
printf("a[%d][%d] = %d\n", i,j, a[i][j] );
}
```

When the above code is compiled and executed, it produces the following result:

```
a[0][0]: 0
a[0][1]: 0
a[1][0]: 1
a[1][1]: 2
a[2][0]: 2
a[2][1]: 4
a[3][0]: 3
a[3][1]: 6
a[4][0]: 4
a[4][1]: 8
```

As explained above, you can have arrays with any number of dimensions, although it is likely that most of the arrays you create will be of one or two dimensions.

14.6.Passing Arrays as Function Arguments

If you want to pass a single-dimension array as an argument in a function, you would have to declare function formal parameter in one of following three ways and all three declaration methods produce similar results because each tells the compiler that an integer pointer is going to be received. Similar way you can pass multi-dimensional array as formal parameters.

Way-1

Formal parameters as a pointer as follows. You will study what is pointer in next chapter.

```
void myFunction(int *param)
{
.
.
.
}
```

Way-2

Formal parameters as a sized array as follows:

```
void myFunction(int param[10])
{
.
.
.
}
```

Way-3

Formal parameters as an unsized array as follows:

```
void myFunction(int param[])
{
.
.
.
}
```

Example

Now, consider the following function, which will take an array as an argument along with another argument and based on the passed arguments, it will return average of the numbers passed through the array as follows:

```
double getAverage(int arr[], int size)
{
int i;
double avg;
double sum;
for (i = 0; i < size; ++i)
{
sum += arr[i];
}
avg = sum / size;
return avg;
}
```

Now, let us call the above function as follows:

```
#include <stdio.h>
/* function declaration */
double getAverage(int arr[], int size);
int main ()
{
/* an int array with 5 elements */
int balance[5] = {1000, 2, 3, 17, 50};
double avg;
/* pass pointer to the array as an argument */
avg = getAverage( balance, 5 ) ;
/* output the returned value */
printf( "Average value is: %f ", avg );
return 0;
}
```

When the above code is compiled together and executed, it produces the following result:

```
Average value is: 214.400000
```

As you can see, the length of the array doesn't matter as far as the function is concerned because C performs no bounds checking for the formal parameters.

Return array from function

C programming language does not allow to return an entire array as an argument to a function. However, you can return a pointer to an array by specifying the array's name without an index. You will study pointer in next chapter so you can skip this chapter until you understand the concept of Pointers in C.

If you want to return a single-dimension array from a function, you would have to declare a function returning a pointer as in the following example:

```
int * myFunction()
{
}
```

Second point to remember is that C does not advocate to return the address of a local variable to outside of the function so you would have to define the local variable as static variable.

Now, consider the following function which will generate 10 random numbers and return them using an array and call this function as follows:

```
#include <stdio.h>
/* function to generate and return random numbers */
int * getRandom( )
{
static int r[10];
int i;
/* set the seed */
srand( (unsigned)time( NULL ) );
for ( i = 0; i < 10; ++i)
{
r[i] = rand();
printf( "r[%d] = %d\n", i, r[i]);
}
return r;
}
/* main function to call above defined function */
int main ()
{
/* a pointer to an int */
int *p;
int i;
p = getRandom();
for ( i = 0; i < 10; i++ )
{
printf( "*(p + %d) : %d\n", i, *(p + i));
}
return 0;
}
```

15. Functions in C.

15.1. Introduction

Function is a group of statements that together perform a task. Every C program has at least one function, which is main(), and all the most trivial programs can define additional functions.

You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division usually is so each function performs a specific task.

A function declaration tells the compiler about a function's name, return type, and parameters. A function definition provides the actual body of the function.

Functions

The C standard library provides numerous built-in functions that your program can call. For example, function strcat() to concatenate two strings, function memcpy() to copy one memory location to another location and many more functions.

A function is known with various names like a method or a sub-routine or a procedure, etc.

15.2. Defining a Function.

A function definition in C programming language consists of a function header and a function body. General form of a function definition in C programming language is as follows:

```
return_type function_name( parameter list )
{
body of the function
}
```

Return Type: A function may return a value. The return_type is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword void.

Function Name: This is the actual name of the function. The function name and the parameter list together constitute the function signature.

Parameters: A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.

Function Body: The function body contains a collection of statements that define what the function does.

Example

Following is the source code for a function called max(). This function takes two parameters num1 and num2 and returns the maximum between the two:

```
/* function returning the max between two numbers */
int max(int num1, int num2)
{
/* local variable declaration */
int result;
if (num1 > num2)
result = num1;
else
result = num2;
return result;
}
```

15.3. Function Declarations

A function declaration tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.

A function declaration has the following parts:

return_type function_name(parameter list);

For the above defined function max(), following is the function declaration:

int max(int num1, int num2);

Parameter names are not important in function declaration only their type is required, so following is also valid declaration:

int max(int, int);

Function declaration is required when you define a function in one source file and you call that function in another file. In such case you should declare the function at the top of the file calling the function.

15.4. Calling a Function

While creating a C function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task.

When a program calls a function, program control is transferred to the called function. A called function performs defined task, and when its return statement is executed or when its function-ending closing brace is reached, it returns program control back to the main program.

To call a function, you simply need to pass the required parameters along with function name, and if function returns a value, then you can store returned value. For example:

```
#include <stdio.h>
/* function declaration */
int max(int num1, int num2);
int main ()
{
/* local variable definition */
int a = 100;
int b = 200;
int ret;
/* calling a function to get max value */
ret = max(a, b);
printf( "Max value is : %d\n", ret );
return 0;
}
/* function returning the max between two numbers */
int max(int num1, int num2)
{
/* local variable declaration */
int result;
if (num1 > num2)
result = num1;
else
result = num2;
return result;
}
```

While r

Max value is : 200

Function Arguments

If a function is to use arguments, it must declare variables that accept the values of the arguments. These variables are called the formal parameters of the function.

The formal parameters behave like other local variables inside the function and are created upon entry into the function and destroyed upon exit.

While calling a function, there are **two ways** that arguments can be passed to a function:

15.5 Function call by value

The **call by value** method of passing arguments to a function copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.

By default, C programming language uses call by value method to pass arguments. In general, this means that code within a function cannot alter the arguments used to call the function. Consider the function swap() definition as follows.

```
/* function definition to swap the values */
void swap(int x, int y)
{
int temp;
temp = x; /* save the value of x */
x = y; /* put y into x */
y = temp; /* put x into y */
return;
}
```

Now, let us call the function swap() by passing actual values as in the following example:

```
#include <stdio.h>
/* function declaration */
void swap(int x, int y);
int main ()
{
/* local variable definition */
int a = 100;
int b = 200;
printf("Before swap, value of a : %d\n", a );
printf("Before swap, value of b : %d\n", b );
/* calling a function to swap the values */
swap(a, b);
printf("After swap, value of a : %d\n", a );
printf("After swap, value of b : %d\n", b );
return 0;
}
```

Let us put above code in a single C file, compile and execute it, it will produce the following result:

```
Before swap, value of a :100
Before swap, value of b :200
```

Which shows that there is no change in the values though they had been changed inside the function.

15.6 Function call by reference

The call by reference method of passing arguments to a function copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the passed argument.

To pass the value by reference, argument pointers are passed to the functions just like any other value. So accordingly you need to declare the function parameters as pointer types as in the following function swap(), which exchanges the values of the two integer variables pointed to by its arguments.

```
/* function definition to swap the values */
void swap(int *x, int *y)
{
int temp;
temp = *x; /* save the value at address x */
*x = *y; /* put y into x */
*y = temp; /* put x into y */
return;
}
```

Let us call the function swap() by passing values by reference as in the following example:

```c
#include <stdio.h>
/* function declaration */
void swap(int *x, int *y);
int main ()
{
/* local variable definition */
int a = 100;
int b = 200;
printf("Before swap, value of a : %d\n", a );
printf("Before swap, value of b : %d\n", b );
/* calling a function to swap the values.
* &a indicates pointer to a ie. address of variable a and
* &b indicates pointer to b ie. address of variable b.
*/
swap(&a, &b);
printf("After swap, value of a : %d\n", a );
printf("After swap, value of b : %d\n", b );
return 0;
}
```

Let us put above code in a single C file, compile and execute it, it will produce the following result:

```
Before swap, value of a :100

Before swap, value of b :200

After swap, value of a :200

After swap, value of b :100
```

Which shows that there is no change in the values though they had been changed inside the function.

Summary:

The function is a program that calls another program for execution.
Passing parameters to the functions could be understood using the program.

6.2. Recursive function

Recursion is the process of repeating items in a self-similar way. Same applies in programming languages as well where if a programming allows you to call a function inside the same function that is called **recursive call** of the function as follows.

```
void recursion()
{
recursion(); /* function calls itself */
}
int main()
{
recursion();
}
```

The C programming language supports recursion, i.e., a function to call itself. But while using recursion, programmers need to be careful to define an exit condition from the function, otherwise it will go in infinite loop.

Recursive function are very useful to solve many mathematical problems like to calculate factorial of a number, generating Fibonacci series, etc.

Number Factorial

Following is an example, which calculates factorial for a given number using a recursive function:

```
#include <stdio.h>
int factorial(unsigned int i)
{
if(i <= 1)
{
return 1;
}
return i * factorial(i - 1);
}
int main()

{
int i = 15;
printf("Factorial of %d is %d\n", i, factorial(i));
return 0;
}
```

When the above code is compiled and executed, it produces the following result:

Factorial of 15 is 2004310016

Fibonacci Series

Following is another example, which generates Fibonacci series for a given number using a recursive function:

```
#include <stdio.h>
int fibonaci(int i)
{
if(i == 0)
{
return 0;
}
if(i == 1)
{
return 1;
}
return fibonaci(i-1) + fibonaci(i-2);
}
int main()
{
int i;
for (i = 0; i < 10; i++)
{
printf("%d\t%n", fibonaci(i));
}
return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
0 1 1 2 3 5 8 13 21 34
```

COURSE PLAN

Select a course (a theory subject): **Programming for Problem Solving**

Select a unit: **Module IV**

Select a Topic: **Structure & Union, Strings**

1. **Objectives**

   To study about how to create & use a Structures for heterogeneous data items

2. **Outcomes**

   Students will acquire knowledge on structure and array of structure in C Language

3. **Pre-requisites**

   Knowledge of Data types, Array and Function.

4. **Terminology used other than normal known scientific / engg terms and their fundamental explanations / relations**

   Not Applicable

Plan for the lecture delivery

1. **How to plan for delivery – black board / ppt / animated ppt / (decide which is good for this topic)**

   Power Point Presentation

2. **How to explain the definition and terms with in it**

   Explain the concept of Array and compare with Structure variables.

3. **How to start the need for any derivation / procedure / experiment / case study**

   Not Applicable

4. **Physical meaning of math equations / calculations**

   Not Applicable

5. **Units and their physical meaning to make them understand practical reality and comparison between different units**

   Not Applicable

6. **What is the final conclusion?**

   Students will acquire knowledge on structure and array of structure in C Language and Know the various string Operations in C Language.

7. **How to put it in a nut shell**

8. **Important points for understanding / memorizing / make it long lasting**

Structure is a group of Dissimilar items.

String is a Group of Character

Various String functions are a) strlen() b) strcmp() c)strcat d)strlwr e)strupr() f) strrev()

9. **Questions and cross questions in that topic to make them think beyond the topic.**

A 1. Example of heterogeneous data type in C language?

a) Unsigned int b) structure c) array d) int

……………………………

2. Syntax of structure declaration ……………………………

3. Which operator to use …………………to access Structure individual Elements

B 1. What is structure?

2. What is different between structure & array?

3. Uses of structure

4. Write the program for Book information storing structure declaration

C 1. Write the program for 100 Books information insertion & displaying using structure

2. Write the program for student information details using structure

D 1. Union is a _______________item:

2. Different between union & structure is ________________mechanism

3. State True or False:

a) Union allocate more memory

4. Union can access only ____________ at a time

a) One member b)All members c) array of member

E 1. Different between Structure & Union

2. Define union

3. Explain Memory mechanism in Union

10. Final conclusions

To study about how to create & use a Structures for heterogeneous data items add to the clarity of the program

Array(Unit III) handle only all **int**s, or all **float**s or all **char**s at a time. In fact when we handle real world data, we don't usually deal with little atoms of information by themselves—things like integers, characters and such. Instead we deal with entities that are collections of things, each thing having its own attributes, just as the entity we call a 'book' is a collection of things such as title, author, call number, publisher, number of pages, date of publication, etc. As you can see all this data is dissimilar, for example author is a string, whereas number of pages is an integer. For dealing with such collections, C provides a data type called 'structure'. A structure gathers together, different atoms of information that comprise a given entity.

17. Structures

Introduction

Data type is the type of the data, that are going to access within the program. 'C' Supports different data types, each data type may have predefined memory requirement and storage representation.

17.1 Why Use Structures

information and how arrays can hold a number of pieces of information of the same data type. These two data types can handle a great variety of situations. But quite often we deal with entities that are collection of dissimilar data types.

Example Suppose you want to store data about a book.

Table 4.1.1

BOOK	**INFORMATIONS**	**DATA TYPE**
	NAME	Character Array (char [100])
	PRICE	float
	NUMBER OF PAGE	int

Then we can follow two approaches:

- Construct individual arrays, one for storing names, another for storing prices and still another for storing number of pages.

- Use a structure variable.

Table 4.1.2

First Approach	**Second Approach**
This approach no doubt allows you to store names, prices and number of pages. But as you must have realized, it is an unwieldy approach that obscures the fact that you are dealing with a group of characteristics related to a single entity	A structure contains a number of data types grouped together. These data types may or may not be of the same type

main( )
{
char name[3] ;
float price[3] ;
int pages[3], i ;
printf ("\nEnter names, prices and no. of pages of 3 books\n") ;
for (i = 0 ; i <= 2 ; i++)
scanf ("%c %f %d", &name[i], &price[i], &pages[i]);
printf ("\nAnd this is what you entered\n") ;
for (i = 0 ; i <= 2 ; i++)
printf ("%c %f %d\n", name[i], price[i], pages[i]);
}
	main( )
	{
	struct book
	{
	char name ;
	float price ;
	int pages ;
	} ;
	struct book b1, b2, b3 ;
	printf ("\nEnter names, prices & no. of pages of 3 books\n") ;
	scanf ("%c %f %d", &b1.name, &b1.price, &b1.pages) ;
	scanf ("%c %f %d", &b2.name, &b2.price, &b2.pages) ;
	scanf ("%c %f %d", &b3.name, &b3.price, &b3.pages) ;
	printf ("\nAnd this is what you entered") ;
	printf ("\n%c %f %d", b1.name, b1.price, b1.pages) ;
	printf ("\n%c %f %d", b2.name, b2.price, b2.pages) ;
	printf ("\n%c %f %d", b3.name, b3.price, b3.pages) ;
	}

And here is the sample run... Enter names, prices and no. of pages of 3 books A 100.00 354 C 256.50 682 F 233.70 512 And this is what you entered	

```
A    100.000000  354

C    256.500000  682

F    233.700000  512
```

Declaring a Structure

The general form of a structure declaration statement is given below:

```
struct <structure name>

{

structure element 1 ;

structure element 2 ;

structure element 3 ;

......

......
} ;
```

For example program Table 4.1.2

```
struct book

{

char name ;

float price ;

int pages ;

} ;
```

This statement defines a new data type called struct book. Each variable of this data type will consist of a character variable called name, a float variable called price and an integer variable called pages

Need to sets aside space in memory in a structure by using creating

struct book b1, b2, b3

It makes available space to hold all the elements in the structure for Table 4.1.2 name is 1 byte, price is 4 bytes, pages is 2 bytes, So total 7 bytes to allocate for b1, similarly s2, s3 also. Table 4.1.2 structure variable contains 7 bytes each.

Various formats to declaring Structure variables

Table 5.1.3

struct book { char name ; float price ; int pages ; } ; struct book b1, b2, b3 ;	struct book { char name ; float price ; int pages ; } b1, b2, b3 ;	Struct { char name ; float price ; int pages ; } b1, b2, b3 ;

Table 4.1.3 are similar to same in declaring Structure variables

Initialize values Statically in Structure variable similar to an array
Example:
struct book b1 = { "Basic", 130.00, 550 } ;
struct book b2 = { "Physics", 150.80, 800 } ;

Note the following points while declaring a structure type:

- The closing brace in the structure type declaration must be followed by a semicolon.

- It is important to understand that a structure type declaration does not tell the compiler to reserve any space in memory. All a structure declaration does is, it defines the 'form' of the structure.

- Usually structure type declaration appears at the top of the source code file, before any variables or functions are defined

Accessing Structure Elements

Access individual elements of use a dot (.) operator. So to refer to Table 4.1.2 of the structure defined in our sample program we have to use,

b1.pages

Similarly, to refer to price we would use,

b1.price

Note that before the dot there must always be a structure variable and after the dot there must always be a structure element.

How Structure Elements are Stored

Whatever be the elements of a structure, they are always stored in contiguous memory locations. The following program would illustrate this:

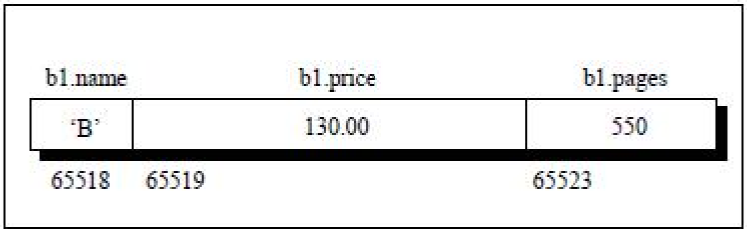
Table 4.1.4

```
/* Memory map of structure elements */
main( )
{
struct book
{
char name ;
float price ;
int pages ;
} ;
struct book b1 = { 'B', 130.00, 550 } ;
printf ( "\nAddress of name = %u", &b1.name ) ;
printf ( "\nAddress of price = %u", &b1.price ) ;

printf ( "\nAddress of pages = %u", &b1.pages ) ;

}
```

Here is the output of the program...

```
Address of name = 65518

Address of price = 65519

Address of pages = 65523
```

Actually the structure elements are stored in memory



17.2 Array of Structures

In our sample program, to store data of 100 books we would be required to use 100 different structure variables from b1 to b100, which is definitely not very convenient. A better approach would be to use an array of structures. Following program shows how to use an array of structures.

```
main( )
{
struct book
{
char name ;
float price ;
int pages ;
} ;
struct book b[100] ;
int i ;
for ( i = 0 ; i <= 99 ; i++ )
{
printf ( "\nEnter name, price and pages " ) ;
scanf ( "%c %f %d", &b[i].name, &b[i].price, &b[i].pages ) ;
}
for ( i = 0 ; i <= 99 ; i++ )
printf ( "\n%c %f %d", b[i].name, b[i].price, b[i].pages ) ;
}
linkfloat( )
{
float a = 0, *b ;
b = &a ; /* cause emulator to be linked */
a = *b ; /* suppress the warning - variable not used */
}
```

Now a few comments about the program:

(a) Notice how the array of structures is declared...

struct book b[100] ;

This provides space in memory for 100 structures of the type struct book.

(b) The syntax we use to reference each element of the **array b** is similar to the syntax used for arrays of **ints and chars**. For example, we refer to zeroth book's price as b[0].price. Similarly, we refer first book's pages as b[1].pages.

(c) It should be appreciated what careful thought Dennis Ritchie has put into C language. He first defined array as a collection of similar elements; then realized that dissimilar data types that are often found in real life cannot be handled using arrays, therefore created a new data type called structure. But even using structures programming convenience could not be achieved, because a lot of variables (b1 to b100 for storing data about hundred books) needed to be handled. Therefore he allowed us to create an array of structures; an array of similar data types which themselves are a collection of dissimilar data types. Hats off to the genius!

(d) In an array of structures all elements of the array are stored in adjacent memory locations. Since each element of this array is a structure, and since all structure elements are always stored in adjacent locations you can very well visualise the arrangement of array of structures in memory. In our example, b[0]'s name, price and pages in memory would be immediately followed by b[1]'s name, price and pages, and so on.

(e) What is the function linkfloat( ) doing here? If you don't define it you are bound to get the error "Floating Point Formats Not Linked" with majority of C Compilers. What causes this error to occur? When parsing our source file, if the compiler encounters a reference to the address of a float, it sets a flag to have the linker link in the floating-point emulator. A floating point emulator is used to manipulate floating point numbers in runtime library functions like

reference to the float is a bit obscure and the compiler does not detect the need for the emulator. The most common is using scanf( ) to read a float in an array of structures as shown in our program. How can we force the formats to be linked? That's where the linkfloat( ) function comes in. It forces linking of the floating-point emulator into an application. There is no need to call this function, just define it anywhere in your program.

17.3 Uses of Structures

The immediate application that comes to the mind is Database Management. That is, to maintain data about employees in an organization, books in a library, items in a store, financial accounting transactions in a company etc. But mind you, use of structures stretches much beyond database management. They can be used for a variety of purposes like:

Changing the size of the cursor

Clearing the contents of the screen

Placing the cursor at an appropriate position on screen

Drawing any graphics shape on the screen

Receiving a key from the keyboard

Checking the memory size of the computer

Finding out the list of equipment attached to the computer

Formatting a floppy

Hiding a file from the directory

Displaying the directory of a disk

Sending the output to printer

Interacting with the mouse

18 INTRODUCTION TO UNION

INTRODUCTION

To define a union, you must use the union statement in very similar was as you did while defining structure. The union statement defines a new data type, with more than one member for your program. Data type is the type of the data, that are going to access within the program. 'C' Supports different data types, each data type may have predefined memory requirement and storage representation.

Defining a Union

Synatx	Example

union [union tag]
{
  member definition;
  member definition;
  ...
  member definition;
} [one or more union variables];
	union Data
	{
	  int i;
	  float f;
	  char str[20];
	} d1;

Example

```
#include <stdio.h>
#include <string.h>
union student
{
          char name[20];
          char subject[20];
          float percentage;
};
int main()
```

```
{
        union student record1;
        union student record2;
        strcpy(record1.name, "Raju");
        strcpy(record1.subject, "Maths");
        record1.percentage = 86.50;
        printf("Union record1 values example\n");
        printf(" Name       : %s \n", record1.name);
        printf(" Subject    : %s \n", record1.subject);
        printf(" Percentage : %f \n\n", record1.percentage);

    // assigning values to record2 union variable
       printf("Union record2 values example\n");
       strcpy(record2.name, "Mani");
       printf(" Name       : %s \n", record2.name);
       strcpy(record2.subject, "Physics");
       printf(" Subject    : %s \n", record2.subject);
       record2.percentage = 99.50;
       printf(" Percentage : %f \n", record2.percentage);
       return 0;
}
```

```
Output:
Union record1 values example
Name :
Subject :
Percentage : 86.500000
Union record2 values example
Name : Mani
Subject : Physics
Percentage : 99.500000
```

18.1 WHY USE UNION

C Union is also like structure, i.e. collection of different data types which are grouped together. Each element in a union is called member.

Union and structure in C are same in concepts, except allocating memory for their members.

Structure allocates storage space for all its members separately.

Whereas, Union allocates one common storage space for all its members

We can access only one member of union at a time. We can't access all member values at the same time in union. But, structure can access all member values at the same time. This is because, Union allocates one common storage space for all its members. Where as Structure allocates storage space for all its members separately.

Many union variables can be created in a program and memory will be allocated for each union variable separately.

18.2 DIFFERENT BETWEEN STRUCTURE & UNION

S.no	C Structure	C Union
1	Structure allocates storage space for all its members separately.	Union allocates one common storage space for all its members. Union finds that which of its member needs high storage space over other members and allocates that much space
2	Structure occupies higher memory space.	Union occupies lower memory space over structure.
3	We can access all members of structure at a time.	We can access only one member of union at a time.

4	Structure example:
	struct student
	{
	int mark;
	char name[6];
	double average;
	};
		Union example:
		union student
		{
		int mark;
		char name[6];
		double average;
		};

5	For above structure, memory allocation will be like below. int mark – 2B char name[6] – 6B double average – 8B Total memory allocation = 2+6+8 = 16 Bytes	For above union, only 8 bytes of memory will be allocated since double data type will occupy maximum space of memory over other data types. Total memory allocation = 8 Bytes

20 STRINGS

INTRODUCTION

The way a group of integers can be stored in an integer array, similarly a group of characters can be stored in a character array. Character arrays are many a time also called strings.Character arrays or strings are used by programming languages to manipulate text such as words and sentences.

20.1 What are Strings

A string constant is a one-dimensional array of characters terminated by a null ('\0')
For example, char name[ ] = { 'H', 'A', 'E', 'S', 'L', 'E', 'R', '\0' } ;
Each character in the array occupies one byte of memory and the last character is always '\0'.
ASCII value of '\0' is 0, show the way a character array is stored in memory.
Note that the elements of the character array are stored in contiguous memory locations.
The terminating null ('\0') is important, because it is the only way the functions that work with a string can know where the string ends

65525	\0
65524	r
65523	e
65522	l
65521	s
65519	e
65520	h
65518	a

C concedes the fact that you would use strings very often and hence provides a **shortcut for initializing strings**. For example, the string used above can also be initialized as,
char name[ ] = "HAESLER" ;

Note that, in this declaration '\0' is not necessary. C inserts the null character automatically.

/* Program to demonstrate printing of a string */
main( )
{
char name[ ] = "Klinsman" ;
int i = 0 ;
while (i <= 7)
{
printf ("%c", name[i]) ;
i++ ;
}}
	/* Program to demonstrate printing of a string */
	main( )
	{
	char name[ ] = "Klinsman" ;
	char *ptr ;
	ptr = name ; /* store base address of string */
	while (*ptr != `\0')
	{
	printf ("%c", *ptr) ;
	ptr++ ;
	}}

And here is the output... Klinsman	

As with the integer array, by mentioning the name of the array we get the base address (address of the zeroth element) of the array. This base address is stored in the variable ptr using,

ptr = name ;

Once the base address is obtained in ptr, *ptr would yield the value at this address, which gets printed promptly through,

printf ("%c", *ptr) ;

Then, ptr is incremented to point to the next character in the string. This derives from two facts: array elements are stored in contiguous memory locations and on incrementing a pointer it points to the immediately next location of its type. This process is carried out till ptr doesn't point to the last character in the string, that is, '\0'.

In fact, the character array elements can be accessed exactly in the same way as the elements of an integer array. Thus, all the following notations refer to the same element:

name[i]
***(name + i)**
***(i + name)**
i[name]

The %s used in printf( ) is a format specification for printing out a string.

```
main( )
{
char name[25] ;
printf ( "Enter your name " ) ;
scanf ( "%s", name ) ;
printf ( "Hello %s!", name ) ;
}
```

And here is a sample run of the program...

```
Enter your name Debashish
Hello Debashish!
```

Note that the declaration char name[25] sets aside 25 bytes under the array name[ ], whereas the scanf( ) function fills in the characters typed at keyboard into this array until the enter key is hit.

Once enter is hit, scanf( ) places a '\0' in the array. Naturally, we should pass the base address of the array to the scanf( ) function.

While entering the string using scanf( ) we must be cautious about two things:

The length of the string should not exceed the dimension of the character array

This is because the C compiler doesn't perform bounds checking on character arrays. Hence, if you carelessly exceed the bounds there is always a danger of overwriting something important, and in that event, you would have nobody to blame but yourselves.

scanf( ) is not capable of receiving multi-word strings.

Therefore names such as 'Debashish Roy' would be unacceptable. The way to get around this limitation is by using the function gets( ). The usage of functions gets( ) and its counterpart puts( ) is shown below.

```
main( )
{
char name[25] ;
printf ( "Enter your full name " ) ;
gets ( name ) ;
puts ( "Hello!" ) ;
puts ( name ) ;
}
```

And here is the output...

```
Enter your name Debashish Roy
Hello!
Debashish Roy
```

On displaying a string, unlike printf( ), puts( ) places the cursor on the next line. Though gets( ) is capable of receiving only one string at a time, the plus point with gets( ) is that it can receive a multi-word string.

20.2 Pointers and Strings

Suppose we wish to store “Hello”. We may either store it in a string or we may ask the C compiler to store it at some location in memory and assign the address of the string in a char pointer. This is shown below:

```
char str[ ] = "Hello" ;
char *p = "Hello" ;
```

There is a subtle difference in usage of these two forms. For example, we cannot assign a string to another, whereas, we can assign a char pointer to another char pointer. This is shown in the following program.

```
main( )
{
char str1[ ] = "Hello" ;
char str2[10] ;
char *s = "Good Morning" ;
char *q ;
str2 = str1 ; /* error */
q = s ; /* works */
}
```

Also, once a string has been defined it cannot be initialized to another set of characters. Unlike strings, such an operation is perfectly valid with char pointers.

```
main( )
{
char str1[ ] = "Hello" ;
char *p = "Hello" ;
str1 = "Bye" ; /* error */
p = "Bye" ; /* works */
}
```

20.3 Standard Library String Functions

With every C compiler a large set of useful string handling library functions are provided.

Function	**Use**
strlen	Finds length of a string
strlwr	Converts a string to lowercase
strupr	Converts a string to uppercase
strcat	Appends one string at the end of another
strncat	Appends first n characters of a string at the end of another
strcpy	Copies a string into another
strncpy	Copies first n characters of one string into another
strcmp	Compares two strings
strncmp	Compares first n characters of two strings
strcmpi	Compares two strings without regard to case ("i" denotes that this function ignores case)
stricmp	Compares two strings without regard to case (identical to strcmpi)
strnicmp	Compares first n characters of two strings without regard to case
strdup	Duplicates a string
strchr	Finds first occurrence of a given character in a string
strrchr	Finds last occurrence of a given character in a string
strstr	Finds first occurrence of a given string in another string
strset	Sets all characters of string to a given character
strnset	Sets first n characters of a string to a given character
strrev	Reverses string

strlen( )

This function counts the number of characters present in a string. Its usage is illustrated in the following program.

main( )
{
char arr[ ] = "Bamboozled" ;
int len1, len2 ;
len1 = strlen (arr) ;
len2 = strlen ("Humpty Dumpty") ;
printf ("\nstring = %s length = %d", arr, len1) ;
printf ("\nstring = %s length = %d", "Humpty Dumpty", len2) ;
}
	Description
	
	Note that in the first call to the function strlen( ), we are passing the base address of the string, and the function in turn returns the length of the string. While calculating the length it doesn't count '\0'.
	
	Even in the second call,
	len2 = strlen ("Humpty Dumpty") ;
	what gets passed to strlen( ) is the address of the string and not the string itself

The output would be...
string = Bamboozled length = 10
string = Humpty Dumpty length = 13	

strcpy( )

This function copies the contents of one string into another. The base addresses of the source and target strings should be supplied to this function.

main( )
{
char source[ ] = "Sayonara" ;
char target[20] ;
strcpy (target, source) ;
printf ("\nsource string = %s", source) ;
printf ("\ntarget string = %s", target) ;
}
	Description
	On supplying the base addresses, strcpy( ) goes on copying the characters in source string into the target string till it doesn't encounter the end of source string ('\0'). It is our responsibility to see to it that the target string's dimension is big enough to hold the string being copied into it. Thus, a string gets copied into another, piece-meal, character by character

And here is the output... source string = Sayonara target string = Sayonara	

strcat( )

This function concatenates the source string at the end of the target string. For example, “Bombay” and “Nagpur” on concatenation would result into a string “BombayNagpur”.

main( )
{
char source[ ] = "Folks!" ;
char target[30] = "Hello" ;
strcat (target, source) ;
printf ("\nsource string = %s", source) ;
printf ("\ntarget string = %s", target) ;
}
	Note that the target string has been made big enough to hold the final string

And here is the output... source string = Folks! target string = HelloFolks!	

strcmp( )

This is a function which compares two strings to find out whether they are same or different. The two strings are compared character by character until there is a mismatch or end of one of the strings is reached, whichever occurs first. If the two strings are identical, strcmp( ) returns a value zero. If they’re not, it returns the numeric difference between the ASCII values of the first non-matching pairs of characters. Here is a program which puts strcmp( ) in action.

main( )
{
char string1[ ] = "Jerry" ;
char string2[ ] = "Ferry" ;
int i, j, k ;
i = strcmp (string1, "Jerry") ;
j = strcmp (string1, string2) ;
k = strcmp (string1, "Jerry boy") ;
printf ("\n%d %d %d", i, j, k) ;
}
	In the first call to strcmp( ), the two strings are identical—“Jerry” and “Jerry”—and the value returned by strcmp( ) is zero. In the second call, the first character of “Jerry” doesn't match with the first character of “Ferry” and the result is 4, which is the numeric difference between ASCII value of ‘J’ and ASCII value of ‘F’. In the third call to strcmp( ) “Jerry” doesn’t match with “Jerry boy”, because the null character at the end of “Jerry” doesn’t match the blank in “Jerry boy”. The value returned is -32, which is the value of null character minus the ASCII value of space, i.e., ‘\0’ minus ‘ ’, which is equal to -32.
	The exact value of mismatch will rarely concern us. All we usually want to know is whether or not the first string is alphabetically before the second string. If it is, a negative value is returned; if it isn’t, a positive value is returned. Any non-zero value means there is a mismatch

And here is the output... 0 4 -32	

Summary:

- A string is nothing but an array of characters terminated by '\0'.
- Being an array, all the characters of a string are stored in contiguous memory locations.
- Though **scanf( )** can be used to receive multi-word strings, **gets( )** can do the same job in a cleaner way. Both **printf( )** and **puts( )** can handle multi-word strings.
- Strings can be operated upon using several standard library functions like **strlen( )**, **strcpy( )**, **strcat( )** and **strcmp( )** which can manipulate strings.
- More importantly we imitated some of these functions to learn how these standard library functions are written.
- Though in principle a 2-D array can be used to handle several strings, in practice an array of pointers to strings is preferred since it takes less space and is efficient in processing strings.
- **malloc( )** function can be used to allocate space in memory on the fly during execution of the program.

Questions:

A 1. State True or False

a) String is a an Array

2. "A" is a ___________ while 'A' is a ____________.
3. A string is terminated by a ______ character, which is written as ______.
4. The array **char name[10]** can consist of a maximum of ______ characters.
5. The array elements are always stored in _________ memory locations.

B 1. Write a program that converts all lowercase characters in a given string to its equivalent uppercase character

2. Write any 5 string library function
3. What is String? Explain

C 1. Write a program to sort a set of names stored in an array in alphabetical order.

2. Write a program any eight string library function in brief example.

COURSE PLAN

Select a course (a theory subject): **Programming for Problem Solving**

Select a unit: **Module V**

Select a Topic: **Pointers & Files**

1. **Objectives**

   To study about File handing Pointer in C language

2. **Outcomes**

   Students will acquire knowledge on Pointers and Files in C language

3. **Pre-requisites**

   Basic C Programming

4. **Terminology used other than normal known scientific / engg terms and their fundamental explanations / relations**

   Not Applicable

Plan for the lecture delivery

1. **How to plan for delivery – black board / ppt / animated ppt / (decide which is good for this topic)**

   Power Point Presentation

2. **How to explain the definition and terms with in it**

   Defining the Pointer variable , Defining the files and various file operations and modes.

3. **How to start the need for any derivation / procedure / experiment / case study**

   Not Applicable

4. **Physical meaning of math equations / calculations**

   Not Applicable

5. **Units and their physical meaning to make them understand practical reality and comparison between different units**

   Not Applicable

6. **What is the final conclusion?**

Understand the concept of Pointers and Various File Operations in C Language

7. **How to put it in a nut shell**

Pointer , File, EOF, BOF, Read Mode, Write Mode)

8. **Important points for understanding / memorizing / make it long lasting**

Pointer

- Pointers are variables which hold addresses of other variables.
- Pointers can be used to make a function return more than one value simultaneously.
- The array variable acts as a pointer to the zeroth element of the array. In a 1-D array, zeroth element is a single value, whereas, in a 2-D array this element is a 1-D array.
- On incrementing a pointer it points to the next location of its type.
- Array elements are stored in contiguous memory locations and so they can be accessed using pointers.
- Only limited arithmetic can be done on pointers

Files

1. File I/O can be performed on a character by character basis, a line by line basis, a record by record basis or a chunk by chunk basis.
2. Different operations that can be performed on a file are creation of a new file, opening an existing file, reading from a file, writing to a file, moving to a specific location in a file (seeking and closing file.
3. File I/O is done using a buffer to improve the efficiency.
4. A file can be a text file or a binary file depending upon its contents.
5. Library function convert \n to \r\n or vice versa while writing/reading to/from a file.
6. Many library functions convert a number to a numeric string before writing it to a file, thereby using more space on disk. This can be avoided using functions fread() and fwrite().

   In low level file I/O we can do the buffer management ourselves.

9. **Questions and cross questions in that topic to make them think beyond the topic.**

A 1. Which of these are reasons for using pointers?

a. To manipulate parts of an array
b. To refer to keywords such as for and if
c. To return more than one value from a function
d. To refer to particular programs more conveniently

B 1. What is pointer?
2. Explains the pointer notation
3. Restriction using pointer arithmetic

4. write program for given elements to an Pointer array.(int i=5, j=6,k=7,z=20)

C 1. Find the smallest number in an array using pointers.
2. Explain pointer arithmetic With suitable example?

D 1. What is File
2. What are the various mode to open the file
3. Write a Program to read the content of a File.

E 1. How to handle error during I/O operations in file handling?
2. What are the type of files?

F 1. Develop a C program to read a data file and display its contents in a neat format?
2. Explain about the file read, write and append operations in C with suitable example?

10. Final conclusions

Understand the concept of Pointers and Various File Operations in C Language

19 Pointer

19.1 An Introduction to Pointers & Pointer Notation

- C Pointer is a variable that stores/points the address of another variable.
- C Pointer is used to allocate memory dynamically i.e. at run time.
- The variable might be any of the data type such as int, float, char, double, short etc.

Syntax : data_type *var_name;
Example : int *p; char *p;

- Where, * is used to denote that "p" is pointer variable and not a normal variable.

Pointer Notation

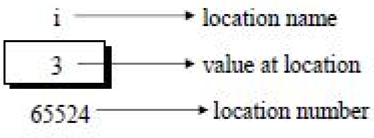
Consider the declaration,

int i = 3 ;

This declaration tells the C compiler to:

(a) Reserve space in memory to hold the integer value.
(b) Associate the name i with this memory location.
(c) Store the value 3 at this location.

We may represent i's location in memory by the following memory map



Key points to remember about pointers in C:

- Normal variable stores the value whereas pointer variable stores the address of the variable.
- The content of the C pointer always be a whole number i.e. address.
- Always C pointer is initialized to null, i.e. int *p = null.
- The value of null pointer is 0.
- & symbol is used to get the address of the variable.
- * symbol is used to get the value of the variable that the pointer is pointing to.
- If pointer is assigned to NULL, it means it is pointing to nothing.
- Two pointers can be subtracted to know how many elements are available between these two pointers.
- But, Pointer addition, multiplication, division are not allowed.
- The size of any pointer is 2 byte (for 16 bit compiler).

```
main( )
{
int i = 3 ;
printf ( "\nAddress of i = %u", &i ) ;
printf ( "\nValue of i = %d", i ) ;
}
```

```
The output of the above program would be:
Address of i = 65524
Value of i = 3
```

Look at the first printf( ) statement carefully. '&' used in this statement is C's 'address of' operator. The expression &i returns the address of the variable i, which in this case happens to be 65524. Since 65524 represents an address, there is no question of a sign being associated with it. Hence it is printed out using %u, which is a format specifier for printing an unsigned integer. We have been using the '&' operator all the time in the scanf( ) statement.

The other pointer operator available in C is '*****', called 'value at address' operator. It gives the value stored at a particular address. The 'value at address' operator is also called 'indirection' operator.

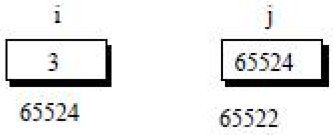
```
main( )
{
int i = 3 ;
printf ( "\nAddress of i = %u", &i ) ;
printf ( "\nValue of i = %d", i ) ;
printf ( "\nValue of i = %d", *( &i ) ) ;
}
```

```
The output of the above program would be:
Address of i = 65524
Value of i = 3
Value of i = 3
```

Note that printing the value of *(&i) is same as printing the value of i.

The expression &i gives the address of the variable i. This address can be collected in a variable, by saying,

j = &i ;



```
main( )
{
int i = 3 ;
int *j ;
j = &i ;
printf ( "\nAddress of i = %u", &i ) ;
printf ( "\nAddress of i = %u", j ) ;
printf ( "\nAddress of j = %u", &j ) ;
printf ( "\nValue of j = %u", j ) ;
printf ( "\nValue of i = %d", i ) ;
printf ( "\nValue of i = %d", *( &i ) ) ;
printf ( "\nValue of i = %d", *j ) ;
}
```

The output of the above program would be:

```
Address of i = 65524
Address of i = 65524
Address of j = 65522
Value of j = 65524
Value of i = 3
Value of i = 3
Value of i = 3
```

Work through the above program carefully, taking help of the memory locations of **i** and **j** shown earlier. This program summarizes everything that we have discussed so far. If you don’t understand the program’s output, or the meanings of **&i**, **&j**, ***j** and ***(&i)**, re-read the last few pages. Everything we say about C pointers from here onwards will depend on your understanding these expressions thoroughly.

19.2 Pointer arithmetic.

```
main( )
 {
int i = 3, *x ;
float j = 1.5, *y ;
char k = 'c', *z ;
printf ( "\nValue of i = %d", i ) ;
 printf ( "\nValue of j = %f", j ) ;
 printf ( "\nValue of k = %c", k ) ;
x = &i ;
 y = &j ;
z = &k ;
printf ( "\nOriginal address in x = %u", x ) ;
printf ( "\nOriginal address in y = %u", y ) ;
printf ( "\nOriginal address in z = %u", z ) ;
x++ ;
y++ ;
z++ ;
printf ( "\nNew address in x = %u", x ) ;
printf ( "\nNew address in y = %u", y ) ;
printf ( "\nNew address in z = %u", z ) ;
}
```

Here is the output of the program.

```
Value of i = 3
Value of j = 1.500000
Value of k = c
Original address in x = 65524
Original address in y = 65520
Original address in z = 65519
New address in x = 65526
New address in y = 65524
New address in z = 65520
```

i	j	k	x	y	z
3	1.500000	‘c’	65524	65520	65519
65524	65520	65519			

Observe the last three lines of the output. 65526 is original value in x plus 2, 65524 is original value in y plus 4, and 65520 is original value in z plus 1.

This so happens because every time a pointer is incremented it points to the immediately next location of its type.

That is why, when the integer pointer x is incremented, it points to an address two locations after the current location, since an int is always 2 bytes long (under Windows/Linux since int is 4 bytes long, new value of x would be 65528).

Similarly, y points to an address 4 locations after the current location and z points 1 location after the current location.

This is a very important result and can be effectively used while passing the entire array to a function.

The way a pointer can be incremented, it can be decremented as well, to point to earlier locations. Thus, the following operations can be performed on a pointer:

(a) For example,

Addition of a number to a pointer.	Subtraction of a number from a pointer

int i = 4, *j, *k ;
j = &i ;
j = j + 1 ;
j = j + 9 ;
k = j + 3 ;
	int i = 4, *j, *k ;
	j = &i ;
	j = j - 2 ;
	j = j - 5 ;
	k = j - 6 ;

pointers... they would never work out.

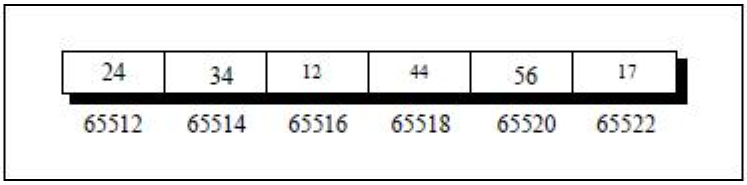
(a) Addition of two pointers
(b) Multiplication of a pointer with a constant
(c) Division of a pointer with a constant

19.3 Pointer to an Array

Now we will try to correlate the following two facts, which we have learnt above:

(a) Array elements are always stored in contiguous memory locations.
(b) A pointer when incremented always points to an immediately next location of its type.

Suppose we have an array **num[ ]** = { 24, 34, 12, 44, 56, 17 }. The following figure shows how this array is located in memory.



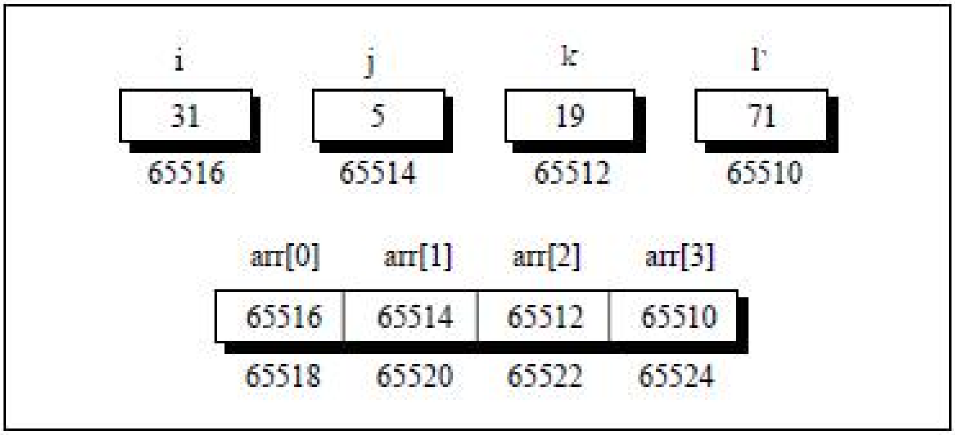
main( )
 {
int arr[ ] = { 10, 20, 36, 72, 45, 36 } ;
int *j, *k ;
j = &arr [4] ;
 k = (arr + 4) ;
if (j == k)
printf ("The two pointers point to the same location") ;
else
 printf ("The two pointers do not point to the same location") ;
}
	main( )
	 {
	int num[ ] = { 24, 34, 12, 44, 56, 17 } ;
	 int i ;
	for (i = 0 ; i <= 5 ; i++)
	{
	printf ("\nelement no. %d ", i) ;
	printf ("address = %u", &num[i]) ;
	 }
	}

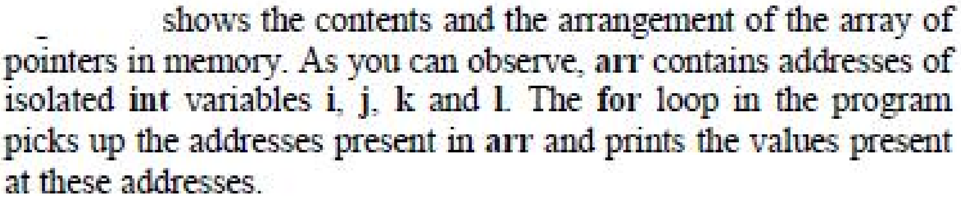
	The output of this program would look like this: element no. 0 address = 65512 element no. 1 address = 65514 element no. 2 address = 65516 element no. 3 address = 65518 element no. 4 address = 65520 element no. 5 address = 65522

19.4 Array of Pointer

The way there can be an array of ints or an array of floats, similarly there can be an array of pointers. Since a pointer variable always contains an address, an array of pointers would be nothing but a collection of addresses. The addresses present in the array of pointers can be addresses of isolated variables or addresses of array elements or any other addresses.

```
main( )
{
int *arr[4] ; /* array of integer pointers */
int i = 31, j = 5, k = 19, l = 71, m ;
arr[0] = &i ;
 arr[1] = &j ;
arr[2] = &k ;
arr[3] = &l ;
for ( m = 0 ; m <= 3 ; m++ )

 printf ( "%d ", * ( arr[m] ) ) ;

}
```




shows the contents and the arrangement of the array of pointers in memory. As you can observe, arr contains addresses of isolated int variables i, j, k and l. The for loop in the program picks up the addresses present in arr and prints the values present at these addresses.

23 FILES

23.1 What is a File?

Abstractly, a file is a collection of bytes stored on a secondary storage device, which is generally a disk of some kind. The collection of bytes may be interpreted, for example, as characters, words, lines, paragraphs and pages from a textual document; fields and records belonging to a database; or pixels from a graphical image. The meaning attached to a particular file is determined entirely by the data structures and operations used by a program to process the file. It is conceivable (and it sometimes happens) that a graphics file will be read and displayed by a program designed to process textual data. The result is that no meaningful output occurs (probably) and this is to be expected. A file is simply a machine decipherable storage media where programs and data are stored for machine usage.

Essentially there are two kinds of files that programmers deal with text files and binary files. These two classes of files will be discussed in the following sections.

ASCII Text files

A text file can be a stream of characters that a computer can process sequentially. It is not only processed sequentially but only in forward direction. For this reason a text file is usually opened for only one kind of operation (reading, writing, or appending) at any given time.

Similarly, since text files only process characters, they can only read or write data one character at a time. (In C Programming Language, Functions are provided that deal with lines of text, but these still essentially process data one character at a time.) A text stream in C is a special kind of file. Depending on the requirements of the operating system, newline characters may be converted to or from carriage-return/linefeed combinations depending on whether data is being written to, or read from, the file. Other character conversions may also occur to satisfy the storage requirements of the operating system. These translations occur transparently and they occur because the programmer has signalled the intention to process a text file.

Binary files

A binary file is no different to a text file. It is a collection of bytes. In C Programming Language a byte and a character are equivalent. Hence a binary file is also referred to as a character stream, but there are two essential differences.

1. No special processing of the data occurs and each byte of data is transferred to or from the disk unprocessed.
2. C Programming Language places no constructs on the file, and it may be read from, or written to, in any manner chosen by the programmer.

Binary files can be either processed sequentially or, depending on the needs of the application, they can be processed using random access techniques. In C Programming Language, processing a file using random access techniques involves moving the current file position to an appropriate place in the file before reading or writing data. This indicates a second characteristic of binary files. They a generally processed using read and write operations simultaneously.

For example, a database file will be created and processed as a binary file. A record update operation will involve locating the appropriate record, reading the record into memory, modifying it in some way, and finally writing the record back to disk at its appropriate location in the file. These kinds of operations are common to many binary files, but are rarely found in applications that process text files.

Why files are needed?

When the program is terminated, the entire data is lost in C programming. If you want to keep large volume of data, it is time consuming to enter the entire data. But, if file is created, these information can be accessed using few commands. There are large numbers of functions to handle file I/O in C language.

23.2 File operations

There are different operations that can be carried out on a file. These are:

a) Creation of new file

b) Opening an existing file

c) Reading from a file

d) Writing to a file

e) Moving to a specific location in a file (seeking)

f) Close a file.

23.3 File opening modes

When you open a file, you must specify how it is to be opened. This means whether to create it from new, overwrite it and whether it's text or binary, read or write and if you want to append to it. This is done using one or more file mode specifiers which are single letters "r", "b", "w", "a" and + (in combination with the other letters). "r" - Opens the file for reading. This fails if the file does not exist or cannot be found. "w" - Opens the file as an empty file for writing. If the file exists, its contents are destroyed. "a" - Opens the file for writing at the end of the file (appending) without removing the EOF marker before writing new data to the file; this creates the file first if it doesn't exist. Adding + to the file mode creates three new modes:

"r+" Opens the file for both reading and writing. (The file must exist.) "w+" Opens the file as an empty file for both reading and writing. If the file exists, its contents are destroyed.
"a+" Opens the file for reading and appending; the appending operation includes the removal of the EOF marker before new data is written to the file and the EOF marker is restored after writing is complete; creates the file first if it doesn't exist.

Opening Modes in Standard I/O

File Mode	Meaning of Mode	During Inexistence of file
r	Open for reading.	If the file does not exist, fopen() returns NULL.
w	Open for writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a	Open for append. i.e, Data is added to end of file.	If the file does not exists, it will be created.
r+	Open for both reading and writing.	If the file does not exist, fopen() returns NULL.
w+	Open for both reading and writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a+	Open for both reading and appending.	If the file does not exists, it will be created.

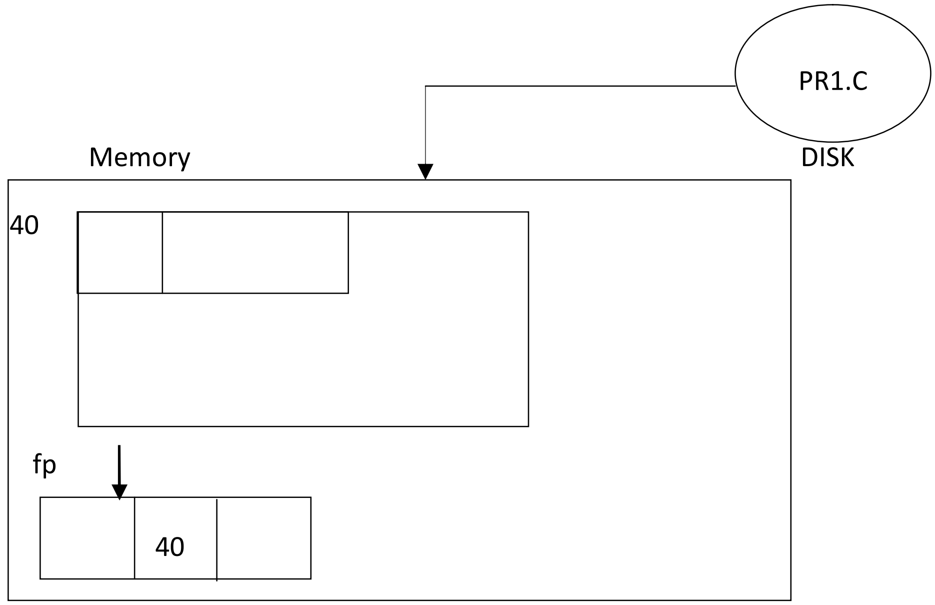
File operations

Opening a File

Before we can read(or write) information from (to) a file on a disk we must open the file. To open the file we have called the function **fopen()**. It would open a file a file "PRC1.C" in 'read mode, which tells the C compiler that we would be reading the contents of the file. Not that "r" is a string not a character; hence the double quotes and not single quotes. In fact **fopen()** performs three important tasks when you open the file in "r' mode:

a) Firstly it searches on the disk the file to be opened.
b) Then it loads the file from the disk into a place in memory called buffer.
c) It sets up a character pointer that points to the first character of the buffer.

Why do we need a buffer at all? Imagine how inefficient it would be to actually access the disk every time we want to read a character from it. Every time we read some think from disk, it takes some time for the disk drive to position the read/write head correctly. On a floppy disk system, the drive motor has to actually start rotating the disk from a standstill position every time the disk is accessed. If this were to be done for every character we read from the disk, it would take a long time to complete the reading operation. This is where a buffer comes in. it would be more sensible to read the contents of the file into the buffer while opening. This is where a buffer comes in. it would be more sensible to read the contents of the file into the buffer while opening the file and then read the file character by character from the buffer rather than from the disk. This is shown in the below figure.



some argument also applies to writing information in a file. Instead of writing characters in the file on the disk one character at time it would be more efficient to write charters in a buffer and then finally transfer the content from the buffer to the disk.

To be able to successfully read from a file information like mode of opening, size of file, place in the file from where the next read operation would be performed, etc. has to be maintained. Since all this information is inter-related, all of it is gathered together by fopen() in a structure called FILE. fopen() return the address of this structure, which we have collected in the stracture pointer called fp. We have declared fp as

FILE * fp;

The FILE structure has been defined in the header file “stdio.h” (standing for standard input/ouput header file). therefore, it is necessary to #include this file.

23.4 Reading file content.

C program to read a file: This program reads a file entered by the user and displays its contents on the screen, fopen function is used to open a file it returns a pointer to structure FILE. FILE is a predefined structure in stdio.h . If the file is successfully opened then fopen returns a pointer to file and if it is unable to open a file then it returns NULL. fgetc function returns a character which is read from the file and fclose function closes the file. Opening a file means we bring file from disk to ram to perform operations on it. The file must be present in the directory in which the executable file of this code sis present.

```
/ * C programming code to open a file and to print it contents on screen. * /


#include <stdio.h>
#include <stdlib.h>
 int main()
{
   char ch, file_name[25];
   FILE *fp;
    printf("Enter the name of file you wish to see\n");
   gets(file_name);
    fp = fopen(file_name,"r");            /* read mode */
    if( fp == NULL )
   {
      printf("Error while opening the file.\n");
      exit(0);
   }
    printf("The contents of %s file are :\n", file_name);
    while( ( ch = fgetc(fp) ) != EOF )
      printf("%c",ch);
    fclose(fp);
   return 0;
}
```

23.5 Error handling While opening a File

There is a possibility that when we try to open a file using the function fopen(), the file may not be opened. While opening the file in “r” mode, this may happen because the file being opened may not be present on the disk at all. And you obviously cannot read a file that doesn’t exist. Similarly, while opening the file for writing, fopen() may fail due to a number of reasons, like, disk space may be insufficient to create a new file, of the disk may be write protected or the disk is damaged and so on.

```
/* C program to handle Error while opening a file */


#include <stdio.h>
#include <stdlib.h>
 int main()
{
   FILE *fp;
   Fp = fopen (“PRGRAM1.C”, “r”);
   if (fp == NULL)
   {
               puts(“cannot open file”);
               Exit(1);
   }
     return 0;
}
```

23.6 Closing the file

To close a file, use the fclose( ) function. The prototype of this function is:

int fclose(FILE *fp);

The fclose( ) function returns zero on success, or EOF if there is an error in closing the file. This function actually, flushes any data still pending in the buffer to the file, closes the file, and releases any memory used for the file. The EOF is a constant defined in the header file stdio.h.

There are various functions provide by C standard library to read and write a file character by character or in the form of a fixed length string.

23.7 Writing to a File

To write a file, use the the fputs() function.

The fputc() function is similar to the putch() function, in the senses that both output characters. However, putch() function always writes to the VDU, whereas, fputc() write to the file. Which file? The file signified by ft. The writing process continues till all characters from the source file have been written to the target file, following which the while loop terminates.

```
/* C programming code to writes strings to a file using the function fputs() */


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
 int main()
{
   FILE *fp;
  char s[80],.
fp=fopen(“Cprogram.txt”, “w”);
  if(fp==NULL)
   {
      printf("Cannot open file\n");
      exit(1);
   }
    printf("\nenter a few lines of text:”);

    while(strlen(gets(s)>0)
    {
       fputs(s, fp);
       fputs(“\n”, fp);
    }
    fclose(fp);
   return 0;
}
```

23.8 A File-Copy Program

We have already used the function fgetc() which reads characters from a file. Its counterpart is a function called fputc() which writes characters to a file. As a practical use of these character I/O functions, we can copy the contents of one file into another, as demonstrated in the following program. This program takes the contents of a file and copies them into another file, character by character.

```
/* File Copy Program */
#include<stdio.h>
#include<stdlib.h>

int main()
{
        FILE *fs, *ft;
        char ch;
        fs = fopen(“prg1.c”,”r”)
        if( fs == NULL)
        {
                puts(“Cannot open Source file”);
                exit(1)
        }
        ft = fopen(“prg2.c”,”w”);
        if ( ft ==NULL)
        {
                puts(“Cannot open target file”);
                fclose(fs);
                exit(2);
        }
        while(1)
        {
                ch = fgetc(fs);
                if(ch==EOF)
                        break;
                else
                fputc(ch,ft);
        }
        fclose(fs);
        fclose(ft)
        return 0;}
```

Writing to a File

The fputc() function is similar to the putch() function, in the senses that both output characters. However, putch() function always writes to the VDU (Visual Display Unit). Whereas, fputc() writes a to the file. Which file? The file signified by ft. The writing process continues till all characters from the source file have been written to the target file, following which the while loop terminates. In the above file copy program is capable of copying only text files. To copy file with extension .EXE or JPG, we need to open the file in binary mode.