

SWAMI DAYANANDA COLLEGE
OF ARTS & SCIENCE
MANJAKKUDI.

SUBJECT NAME :PROGRAMMING IN C

SUBJECT CODE :16SACCS1

Dr. T. Nagarathinam

Assistant Professor, Department of CS

ALLIED COMPUTER SCIENCE FOR B.Sc. PROGRAMMES

Allied Paper I

Programming in C -16SACCS1

Objective: To impart basic knowledge of Programming Skills in C language.

Unit I

Introduction to Computers and their Applications. Computer System Characteristics – Hardware and Software – Types and Generations of Computers – Introduction to I/O and Storage Devices – Number Systems – Flowcharts – Algorithms.

Unit II

Evaluation and Applications of C Structure of a C programme - Data Types – Declarations – Operators – Expressions – Type Conversions – Built-in Functions – Data Input and Output Control Statements : IF, ELSE – IF, GOTO, SWITCH, WHILE – DO, DO – WHILE, FOR BREAK and CONTINUE.

Unit III

Functions – Defining and Accessing Functions – passing parameters to functions – Arguments – recursive functions – Storage Classes – Arrays : Arrays and functions – Arrays and Strings – String functions – String Manipulations.

Unit IV

Pointers – Pointer Declarations - operations on Pointers – pointers to functions – pointers and strings – pointers and arrays – array of pointers structures – structure and pointers – Unions.

Unit V

Data Files – Opening, Closing and Processing files – files with structures and unions- register variables – Bitwise operations – Macros Preprocessors.

Text Book :

1. Computer Today – S.K. Basandra – Galgotia Publications Unit II – V.
2. Programming in C – E.Balagurusamy – Tata McGraw Hill Publication.

Reference Books :

1. Programming with C - Byron S Gottfried – Schaum's Outline Series, Tata McGraw Hill Publications.
2. The Spirit of C – Mullish Cooper – Schaum's Outline Series – Tata McGraw Hill Publications.
3. Let Us C – Yeswant Kanetkar – BPB Publications.

UNIT-1 INTRODUCTION TO COMPUTER

DEFINITION

A computer is an electronic data processing device, which accepts and stores data input, processes the data input, and generates the output in a required format.

Functionalities of a Computer

Step 1 – Takes data as input.

Step 2 – Stores the data/instructions in its memory and uses them as required.

Step 3 – Processes the data and converts it into useful information.

Step 4 – Generates the output.

Step 5 – Controls all the above four steps.



Advantages of Computers

Following are certain advantages of computers.

High Speed

- Computer is a very fast device.
- It is capable of performing calculation of very large amount of data.
- The computer has units of speed in microsecond, nanosecond, and even the picosecond.
- It can perform millions of calculations in a few seconds as compared to man who will spend many months to perform the same task.

Accuracy

- In addition to being very fast, computers are very accurate.
- The calculations are 100% error free.
- Computers perform all jobs with 100% accuracy provided that the input is correct.

Storage Capability

- Memory is a very important characteristic of computers.

- A computer has much more storage capacity than human beings.
- It can store large amount of data.
- It can store any type of data such as images, videos, text, audio, etc.

Diligence

- Unlike human beings, a computer is free from monotony, tiredness, and lack of concentration.
- It can work continuously without any error and boredom.
- It can perform repeated tasks with the same speed and accuracy.

Versatility

- A computer is a very versatile machine.
- A computer is very flexible in performing the jobs to be done.
- This machine can be used to solve the problems related to various fields.
- At one instance, it may be solving a complex scientific problem and the very next moment it may be playing a card game.

Reliability

- A computer is a reliable machine.
- Modern electronic components have long lives.
- Computers are designed to make maintenance easy.

Automation

- Computer is an automatic machine.
- Automation is the ability to perform a given task automatically. Once the computer receives a program i.e., the program is stored in the computer memory, then the program and instruction can control the program execution without human interaction.

Reduction in Paper Work and Cost

- The use of computers for data processing in an organization leads to reduction in paper work and results in speeding up the process.
- As data in electronic files can be retrieved as and when required, the problem of maintenance of large number of paper files gets reduced.
- Though the initial investment for installing a computer is high, it substantially reduces the cost of each of its transaction.

Disadvantages of Computers

Following are certain disadvantages of computers.

No I.Q.

- A computer is a machine that has no intelligence to perform any task.
- Each instruction has to be given to the computer.
- A computer cannot take any decision on its own.

Dependency

- It functions as per the user's instruction, thus it is fully dependent on humans.

Environment

- The operating environment of the computer should be dust free and suitable.

No Feeling

- Computers have no feelings or emotions.
- It cannot make judgment based on feeling, taste, experience, and knowledge unlike humans.

COMPUTER - APPLICATIONS

In this chapter, we will discuss the application of computers in various fields.

Business

A computer has high speed of calculation, diligence, accuracy, reliability, or versatility which has made it an integrated part in all business organizations.

Computer is used in business organizations for –

- Payroll calculations
- Budgeting
- Sales analysis
- Financial forecasting
- Managing employee database
- Maintenance of stocks, etc.

Banking

Today, banking is almost totally dependent on computers.

Banks provide the following facilities –

- Online accounting facility, which includes checking current balance, making deposits and overdrafts, checking interest charges, shares, and trustee records.
- ATM machines which are completely automated are making it even easier for customers to deal with banks.

Insurance

Insurance companies are keeping all records up-to-date with the help of computers. Insurance companies, finance houses, and stock broking firms are widely using computers for their concerns.

Insurance companies are maintaining a database of all clients with information showing –

- Procedure to continue with policies
- Starting date of the policies
- Next due installment of a policy
- Maturity date
- Interests due
- Survival benefits
- Bonus

Education

The computer helps in providing a lot of facilities in the education system.

- The computer provides a tool in the education system known as CBE (Computer Based Education).
- CBE involves control, delivery, and evaluation of learning.
- Computer education is rapidly increasing the graph of number of computer students.
- There are a number of methods in which educational institutions can use a computer to educate the students.
- It is used to prepare a database about performance of a student and analysis is carried out on this basis.

Marketing

In marketing, uses of the computer are following –

- **Advertising** – With computers, advertising professionals create art and graphics, write and revise copy, and print and disseminate ads with the goal of selling more products.
- **Home Shopping** – Home shopping has been made possible through the use of computerized catalogues that provide access to product information and permit direct entry of orders to be filled by the customers.

Healthcare

Computers have become an important part in hospitals, labs, and dispensaries. They are being used in hospitals to keep the record of patients and medicines. It is also used in scanning and diagnosing different diseases. ECG, EEG, ultrasounds and CT scans, etc. are also done by computerized machines.

Following are some major fields of health care in which computers are used.

- **Diagnostic System** – Computers are used to collect data and identify the cause of illness.
- **Lab-diagnostic System** – All tests can be done and the reports are prepared by computer.
- **Patient Monitoring System** – These are used to check the patient's signs for abnormality such as in Cardiac Arrest, ECG, etc.
- **Pharma Information System** – Computer is used to check drug labels, expiry dates, harmful side effects, etc.
- **Surgery** – Nowadays, computers are also used in performing surgery.

Engineering Design

Computers are widely used for engineering purpose.

One of the major areas is CAD (Computer Aided Design) that provides creation and modification of images. Some of the fields are –

- **Structural Engineering** – Requires stress and strain analysis for design of ships, buildings, budgets, airplanes, etc.
- **Industrial Engineering** – Computers deal with design, implementation, and improvement of integrated systems of people, materials, and equipment.
- **Architectural Engineering** – Computers help in planning towns, designing buildings, determining a range of buildings on a site using both 2D and 3D drawings.

Military

Computers are largely used in defence. Modern tanks, missiles, weapons, etc. Military also employs computerized control systems. Some military areas where a computer has been used are –

- Missile Control
- Military Communication
- Military Operation and Planning
- Smart Weapons

Communication

Communication is a way to convey a message, an idea, a picture, or speech that is received and understood clearly and correctly by the person for whom it is meant. Some main areas in this category are –

- E-mail
- Chatting
- Usenet
- FTP

- Telnet
- Video-conferencing

Government

Computers play an important role in government services. Some major fields in this category are –

- Budgets
- Sales tax department
- Income tax department
- Computation of male/female ratio
- Computerization of voters lists
- Computerization of PAN card
- Weather forecasting

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. Nowadays, generation includes both hardware and software, which together make up an entire computer system.

There are five computer generations known till date. Each generation has been discussed in detail along with their time period and characteristics. In the following table, approximate dates against each generation have been mentioned, which are normally accepted.

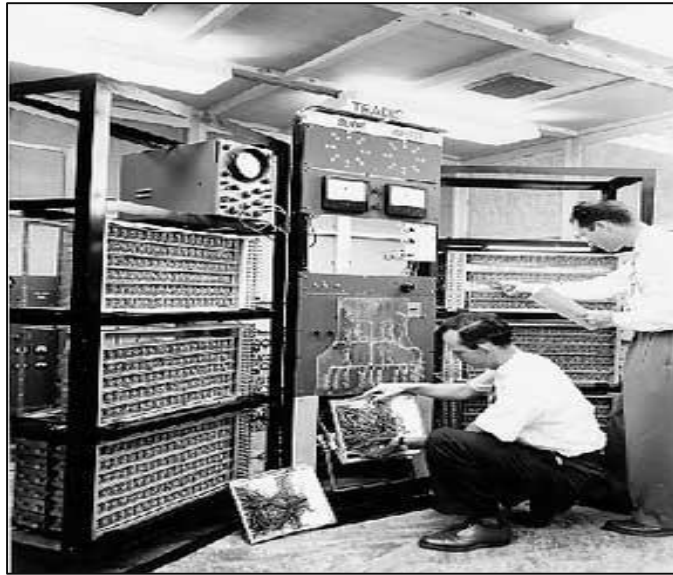
Following are the main five generations of computers.

S.No	Generation & Description	Year
1	<u>First Generation</u> : Vacuum tube based	1946-1959
2	<u>Second Generation</u> : Transistor based	1959-1965
3	<u>Third Generation</u> : Integrated Circuit based	1965-1971
4	<u>Fourth Generation</u> : VLSI microprocessor based	1971-1980
5	<u>Fifth Generation</u> : ULSI microprocessor based	1980- onwards

First Generation Computers

The period of first generation was from 1946-1959. The computers of first generation used 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 the installations used to fuse frequently. Therefore, they were very expensive and only large organizations were able to afford it.

In this generation, mainly batch processing operating system was used. Punch cards, paper tape, and magnetic tape was used as input and output devices. The computers in this generation used machine code as the programming language.



The main features of the first generation are:

- Vacuum tube technology
- Unreliable
- Supported machine language only
- Very costly
- Generated a lot of heat
- Slow input and output devices
- Huge size
- Need of AC
- Non-portable
- Consumed a lot of electricity

Some computers of this generation were:

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

Second Generation Computers

The period of second generation was from 1959-1965. In this generation, transistors were used that were cheaper, consumed less power, more compact in size, more reliable and faster than the first generation machines made of vacuum tubes. In this generation, magnetic cores were used as the primary memory and magnetic tape and magnetic disks as secondary storage devices.

In this generation, assembly language and high-level programming languages like FORTRAN, COBOL were used. The computers used batch processing and multiprogramming operating system.



The main features of second generation are:

- Use of transistors
- Reliable in comparison to first generation computers
- Smaller size as compared to first generation computers
- Generated 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
- AC required
- Supported machine and assembly languages

Some computers of this generation were:

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

Third Generation Computers

The period of third generation was from 1965-1971. The computers of third generation used Integrated Circuits (ICs) in place of transistors. A single IC has many transistors, resistors, and capacitors along with the associated circuitry.

The IC was invented by Jack Kilby. This development made computers smaller in size, reliable, and efficient. In this generation remote processing, time-sharing, multi-programming operating system were used. High-level languages (FORTRAN-II TO IV, COBOL, PASCAL PL/1, BASIC, ALGOL-68 etc.) were used during this generation.



The main features of third generation are:

- IC used
- More reliable in comparison to previous two generations
- Smaller size
- Generated less heat
- Faster
- Lesser maintenance
- Costly

- AC required
- Consumed lesser electricity □ Supported high-level language

Some computers of this generation were:

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

Fourth Generation Computers

The period of fourth generation was from 1971-1980. Computers of fourth generation used Very Large Scale Integrated (VLSI) circuits. VLSI circuits having about 5000 transistors and other circuit elements with 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. In this generation, time sharing, real time networks, distributed operating system were used. All the high-level languages like C, C++, DBASE etc., were used in this generation.



The main features of fourth generation are:

- VLSI technology used
- Very cheap
- Portable and reliable
- Use of PCs
- Very small size
- Pipeline processing

- No AC required
- Concept of internet was introduced
- Great developments in the fields of networks
- Computers became easily available

Some computers of this generation were:

- DEC 10
- STAR 1000
- PDP 11

Fifth Generation Computers

The period of fifth generation is 1980-till date. In the fifth generation, VLSI technology became ULSI (Ultra Large Scale Integration) technology, resulting in the production of microprocessor chips having ten million electronic components.

This generation is based on parallel processing hardware and AI (Artificial Intelligence) software. AI is an emerging branch in computer science, which interprets the means and method of making computers think like human beings. All the high-level languages like C and C++, Java, .Net etc., are used in this generation.



AI includes:

- Robotics
- Neural Networks
- Game Playing
- Development of expert systems to make decisions in real-life situations
- Natural language understanding and 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

COMPUTER - TYPES

Computers can be broadly classified by their speed and computing power.

S.No.	Type	Specifications
1	PC (Personal Computer)	It is a single user computer system having moderately powerful microprocessor
2	Workstation	It is also a single user computer system, similar to personal computer however a more powerful microprocessor has.
3	Mini Computer	It is a multi-user computer system, capable of supporting hundreds of users simultaneously.
4	Main Frame	It is a multi-user computer system, capable of supporting hundreds of users simultaneously. Software technology is different from minicomputer.
5	Supercomputer	It is an extremely fast computer, which can execute hundreds of millions of instructions per second.

PC (Personal Computer)



A PC can be defined as a small, relatively inexpensive computer designed for an individual user. PCs are based on the microprocessor technology that enables manufacturers to put an entire CPU on one chip. Businesses use personal computers for word processing, accounting, desktop publishing, and for running spreadsheet and database management applications. At home, the most popular use for personal computers is playing games and surfing the Internet.

Although personal computers are designed as single-user systems, these systems are normally linked together to form a network. In terms of power, now-a-days high-end models of the Macintosh and PC offer the same computing power and graphics capability as low-end workstations by Sun Microsystems, Hewlett-Packard, and Dell.

Workstation



Workstation is a computer used for engineering applications (CAD/CAM), desktop publishing, software development, and other such types of applications which require a moderate amount of computing power and relatively high quality graphics capabilities.

Workstations generally come with a large, high-resolution graphics screen, large amount of RAM, inbuilt network support, and a graphical user interface. Most workstations also have mass storage device such as a disk drive, but a special type of workstation, called diskless workstation, comes without a disk drive.

Common operating systems for workstations are UNIX and Windows NT. Like PC, workstations are also single-user computers like PC but are typically linked together to form a local-area network, although they can also be used as stand-alone systems.

Minicomputer

It is a midsize multi-processing system capable of supporting up to 250 users simultaneously.



Mainframe

Mainframe is very large in size and is an expensive computer capable of supporting hundreds or even thousands of users simultaneously. Mainframe executes many programs concurrently and supports much simultaneous execution of programs.



Supercomputer

Supercomputers are one of the fastest computers currently available. Supercomputers are very expensive and are employed for specialized applications that require immense amount of mathematical calculations (number crunching).

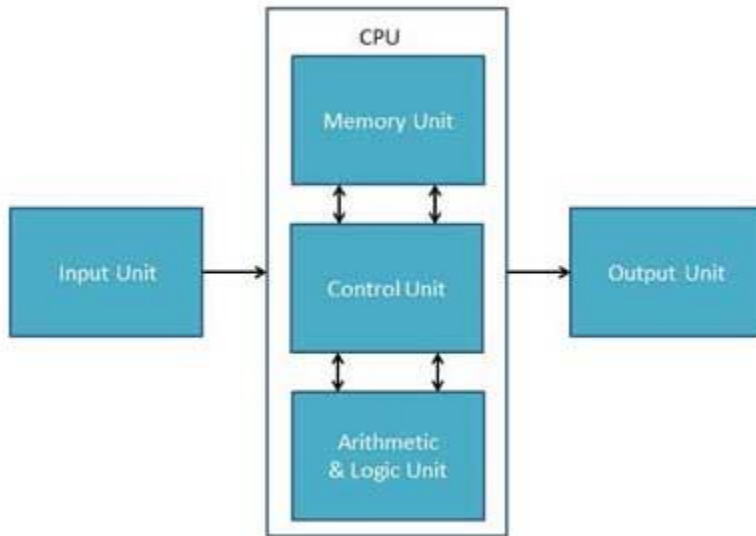


For example, weather forecasting, scientific simulations, (animated) graphics, fluid dynamic calculations, nuclear energy research, electronic design, and analysis of geological data (e.g. in petrochemical prospecting).

Computer - Components

All types of computers follow the same basic logical structure and perform the following five basic operations for converting raw input data into information useful to their users.

S.No.	Operation	Description
1	Take Input	The process of entering data and instructions into the computer system.
2	Store Data	Saving data and instructions so that they are available for processing as and when required.
3	Processing Data	Performing arithmetic, and logical operations on data in order to convert them into useful information.
4	Output Information	The process of producing useful information or results for the user, such as a printed report or visual display.
5	Control the workflow	Directs the manner and sequence in which all of the above operations are performed.



Input Unit

This unit contains devices with the help of which we enter data into the computer. This unit creates a link between the user and the computer. The input devices translate the information into a form understandable by the computer.

CPU (Central Processing Unit)

CPU is considered as the brain of the computer. CPU performs all types of data processing operations. It stores data, intermediate results, and instructions (program). It controls the operation of all parts of the computer.

CPU itself has the following three components –

- ALU (Arithmetic Logic Unit)
- Memory Unit
- Control Unit

Output Unit

The output unit consists of devices with the help of which we get the information from the computer. This unit is a link between the computer and the users. Output devices translate the computer's output into a form understandable by the users.

Computer - CPU(Central Processing Unit)

Central Processing Unit (CPU) consists of the following features –

- CPU is considered as the brain of the computer.
- CPU performs all types of data processing operations.
- It stores data, intermediate results, and instructions (program).
- It controls the operation of all parts of the computer.



CPU itself has following three components.

- Memory or Storage Unit
- Control Unit
- ALU(Arithmetic Logic Unit)

Memory or Storage Unit

This unit can store instructions, data, and intermediate results. This unit supplies information to other units of the computer when needed. It is also known as internal storage unit or the main memory or the primary storage or Random Access Memory (RAM).

Its size affects speed, power, and capability. Primary memory and secondary memory are two types of memories in the computer. Functions of the memory unit are –

- It stores all the data and the instructions required for processing.
- It stores intermediate results of processing.
- It stores the final results of processing before these results are released to an output device.
- All inputs and outputs are transmitted through the main memory.

Control Unit

This unit controls the operations of all parts of the computer but does not carry out any actual data processing operations.

Functions of this unit are –

- It is responsible for controlling the transfer of data and instructions among other units of a computer.
- It manages and coordinates all the units of the computer.
- It obtains the instructions from the memory, interprets them, and directs the operation of the computer.
- It communicates with Input/Output devices for transfer of data or results from storage.
- It does not process or store data.

ALU (Arithmetic Logic Unit)

This unit consists of two subsections namely,

- Arithmetic Section
- Logic Section

Arithmetic Section

Function of arithmetic section is to perform arithmetic operations like addition, subtraction, multiplication, and division. All complex operations are done by making repetitive use of the above operations.

Logic Section

Function of logic section is to perform logic operations such as comparing, selecting, matching, and merging of data.

COMPUTER - INPUT DEVICES

Following are some of the important input devices which are used in a computer –

- 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(OMR)

Keyboard

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



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

The keys on the keyboard are as follows –

S.No	Keys	Description
1	Typing Keys	These keys include the letter keys (A-Z) and digit keys (0-9) which generally give the same layout as that of typewriters.
2	Numeric Keypad	It is used to enter the 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 machines and calculators.
3	Function Keys	The twelve function keys are present on the keyboard which are arranged in a row at the top of the keyboard. Each function key has a unique meaning and is used for some specific purpose.
4	Control keys	These keys provide cursor and screen control. It includes four directional arrow keys. 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 the most popular pointing device. It is a very famous cursor-control device having a small palm size box with a round ball at its base, which senses the movement of the mouse and sends corresponding signals to the CPU when the mouse buttons are pressed.

Generally, it has two buttons called the left and the right button and a wheel is present between the buttons. A mouse can be used to control the position of the cursor on the screen, but it cannot be used to enter text into the computer.



Advantages

- Easy to use
- Not very expensive
- Moves the cursor faster than the arrow keys of the keyboard.

Joystick

Joystick is also a pointing device, which is used to move the 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 the joystick is similar to that of a mouse. It is mainly used in Computer Aided Designing (CAD) and playing computer games.

Light Pen

Light pen is a pointing device 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 the tip of a light pen is moved over the monitor screen and the pen button is pressed, its photocell sensing element detects the screen location and sends the corresponding signal to the CPU.

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 the ball, the 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, or a square.

Scanner

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



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

Digitizer

Digitizer is an input device which converts analog information into digital form. Digitizer can convert a signal from the television or 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 as it converts graphics and pictorial data into binary inputs. A graphic tablet as digitizer is used for fine works of drawing and image manipulation applications.

Microphone

Microphone is an input device to input sound that is then stored in a digital form.



The microphone is used for various applications such as adding sound to a multimedia presentation or for mixing music.

Magnetic Ink Card Reader (MICR)

MICR input device is generally used in banks as there are 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 is that it is fast and less error prone.

Optical Character Reader (OCR)

OCR is an input device used to read a printed text.



OCR scans the text optically, character by character, converts them into a machine readable code, and stores the text on the system memory.

Bar Code Readers

Bar Code Reader is a device used for reading bar coded data (data in the form of light and dark lines). Bar coded data is generally used in labelling goods, numbering the books, etc. It may be a handheld 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 that the bar code reader is connected to.

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.

COMPUTER - OUTPUT DEVICES

Following are some of the important output devices used in a computer.

- Monitors
- Graphic Plotter
- Printer

Monitors

Monitors, commonly called as **Visual Display Unit (VDU)**, are 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 number of pixels.

There are two kinds of viewing screen used for monitors.

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

Cathode-Ray Tube (CRT) Monitor

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



A finite number of characters 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. Most screens are capable of displaying 80 characters of data horizontally and 25 lines vertically.

There are some disadvantages of CRT –

- Large in Size
- High power consumption

Flat-Panel Display Monitor

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



The flat-panel display is divided into two categories –

- **Emissive Displays** – Emissive displays are devices that convert electrical energy into light. For example, plasma panel and LED (Light-Emitting Diodes).
- **Non-Emissive Displays** – Non-emissive displays use optical effects to convert sunlight or light from some other source into graphics patterns. For example, LCD (Liquid-Crystal Device).

Printers

Printer is an output device, which is used to print information on paper.

There are two types of printers –

- Impact Printers
- Non-Impact Printers

Impact Printers

Impact printers print the characters by striking them on the ribbon, which is then pressed on the paper.

Characteristics of Impact Printers are the following –

- Very low consumable costs
- 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 the printers which print one character at a time.

These are further divided into two types:

- Dot Matrix Printer(DMP)
- Daisy Wheel

Dot Matrix Printer

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



Advantages

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

Disadvantages

- Slow Speed
- Poor Quality

Daisy Wheel

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



Advantages

- More reliable than DMP
- Better quality
- Fonts of character can be easily changed

Disadvantages

- Slower than DMP
- Noisy
- More expensive than DMP

Line Printers

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



These are of two types –

- Drum Printer
- Chain Printer

Drum Printer

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

Advantages

- Very high speed

Disadvantages

- Very expensive
- Characters fonts cannot be changed

Chain Printer

In this printer, a chain of character sets is used, hence it is called Chain Printer. A standard character set may have 48, 64, or 96 characters.

Advantages

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

Disadvantages

- Noisy

Non-impact Printers

Non-impact printers print the characters without using the ribbon. These printers print a complete page at a time, thus they are 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
- Supports many fonts and different character size

Laser Printers

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



Advantages

- Very high speed
- Very high quality output
- Good graphics quality
- Supports many fonts and different character size

Disadvantages

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

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. Color 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 the cost per page is high
- Slow as compared to laser printer

COMPUTER - MEMORY

A memory is just like a human brain. It is used to store data and instructions. Computer memory is the storage space in the computer, where data is to be processed and instructions required for processing are stored. The memory is divided into large number of small parts called cells. Each location or cell has a unique address, which varies from zero to memory size minus one. For example, if the computer has 64k words, then this memory unit has $64 * 1024 = 65536$ memory locations. The address of these locations varies from 0 to 65535.

Memory is primarily of three types –

- Cache Memory
- Primary Memory/Main Memory
- Secondary Memory

Cache Memory

Cache memory is a very high speed semiconductor memory which can speed up the CPU. It acts as a buffer between the CPU and the main memory. It is used to hold those parts of data and program which are most frequently used by the CPU. The parts of data and programs are transferred from the disk to cache memory by the operating system, from where the CPU can access them.

Advantages

The advantages of cache memory are as follows –

- Cache memory is faster than main memory.
- It consumes less access time as compared to main memory.
- It stores the program that can be executed within a short period of time.
- It stores data for temporary use.

Disadvantages

The disadvantages of cache memory are as follows –

- Cache memory has limited capacity.
- It is very expensive.

Primary Memory (Main Memory)

Primary memory holds only those data and instructions on which the computer is currently working. It has a limited capacity and data is lost when power is switched off. It is generally made up of semiconductor device. These memories are not as fast as registers. The data and instruction required to be processed resides in the main memory. It is divided into two subcategories RAM and ROM.



Characteristics of Main Memory

- These are semiconductor memories.
- It is known as the main memory.
- Usually volatile memory.
- Data is lost in case power is switched off.
- It is the working memory of the computer.
- Faster than secondary memories.
- A computer cannot run without the primary memory.

Secondary Memory

This type of memory is also known as external memory or non-volatile. It is slower than the main memory. These are used for storing data/information permanently. CPU directly does not access these memories, instead they are accessed via input-output routines. The contents of secondary memories are first transferred to the main memory, and then the CPU can access it. For example, disk, CD-ROM, DVD, etc.



Characteristics of Secondary Memory

- These are magnetic and optical memories.
- It is known as the backup memory.
- It is a non-volatile memory.
- Data is permanently stored even if power is switched off.
- It is used for storage of data in a computer.
- Computer may run without the secondary memory.
- Slower than primary memories.

Random Access Memory

RAM (Random Access Memory) is the internal memory of the CPU for storing data, program, and program result. It is a read/write memory which stores data until the machine is working. As soon as the machine is switched off, data is erased.



Access time in RAM is independent of the address, that is, each storage location inside the memory is as easy to reach as other locations and takes the same amount of time. Data in the RAM can be accessed randomly but it is very expensive.

RAM is volatile, i.e. data stored in it is lost when we switch off the computer or if there is a power failure. Hence, a backup Uninterruptible Power System (UPS) is often used with

computers. RAM is small, both in terms of its physical size and in the amount of data it can hold.

RAM is of two types –

- Static RAM (SRAM)
- Dynamic RAM (DRAM)

Static RAM (SRAM)

The word **static** indicates that the memory retains its contents as long as power is being supplied. However, data is lost when the power gets down due to volatile nature. SRAM chips use a matrix of 6-transistors and no capacitors. Transistors do not require power to prevent leakage, so SRAM need not be refreshed on a regular basis.

There is extra space in the matrix, hence SRAM uses more chips than DRAM for the same amount of storage space, making the manufacturing costs higher. SRAM is thus used as cache memory and has very fast access.

Characteristic of Static RAM

- Long life
- No need to refresh
- Faster
- Used as cache memory
- Large size
- Expensive
- High power consumption

Dynamic RAM (DRAM)

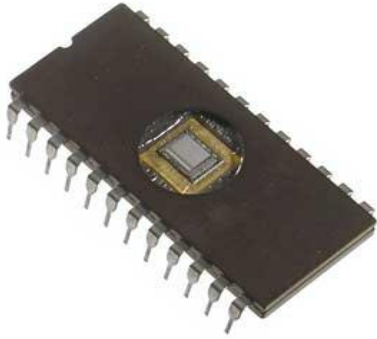
DRAM, unlike SRAM, must be continually **refreshed** in order to maintain the data. This is done by placing the memory on a refresh circuit that rewrites the data several hundred times per second. DRAM is used for most system memory as it is cheap and small. All DRAMs are made up of memory cells, which are composed of one capacitor and one transistor.

Characteristics of Dynamic RAM

- Short data lifetime
- Needs to be refreshed continuously
- Slower as compared to SRAM
- Used as RAM
- Smaller in size
- Less expensive
- Less power consumption

Computer - Read Only Memory

ROM stands for **Read Only Memory**. The memory from which we can only read but cannot write on it. This type of memory is non-volatile. The information is stored permanently in such memories during manufacture. A ROM stores such instructions that are required to start a computer. This operation is referred to as **bootstrap**. ROM chips are not only used in the computer but also in other electronic items like washing machine and microwave oven.



Let us now discuss the various types of ROMs and their characteristics.

MROM (Masked ROM)

The very first ROMs were hard-wired devices that contained a pre-programmed set of data or instructions. These kind of ROMs are known as masked ROMs, which are inexpensive.

PROM (Programmable Read Only Memory)

PROM is read-only memory that can be modified only once by a user. The user buys a blank PROM and enters the desired contents using a PROM program. Inside the PROM chip, there are small fuses which are burnt open during programming. It can be programmed only once and is not erasable.

EPROM (Erasable and Programmable Read Only Memory)

EPROM can be erased by exposing it to ultra-violet light for a duration of up to 40 minutes. Usually, an EPROM eraser achieves this function. During programming, an electrical charge is trapped in an insulated gate region. The charge is retained for more than 10 years because the charge has no leakage path. For erasing this charge, ultra-violet light is passed through a quartz crystal window (lid). This exposure to ultra-violet light dissipates the charge. During normal use, the quartz lid is sealed with a sticker.

EEPROM (Electrically Erasable and Programmable Read Only Memory)

EEPROM is programmed and erased electrically. It can be erased and reprogrammed about ten thousand times. Both erasing and programming take about 4 to 10 ms (millisecond). In EEPROM, any location can be selectively erased and programmed. EEPROMs can be erased one byte at a time, rather than erasing the entire chip. Hence, the process of reprogramming is flexible but slow.

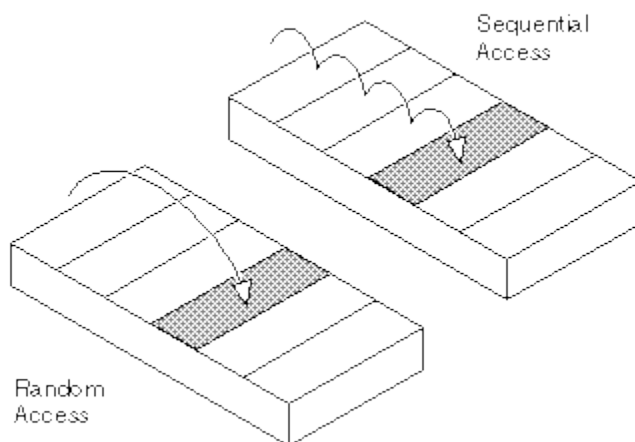
Advantages of ROM

The advantages of ROM are as follows –

- Non-volatile in nature
- Cannot be accidentally changed
- Cheaper than RAMs
- Easy to test
- More reliable than RAMs
- Static and do not require refreshing
- Contents are always known and can be verified

AUXILIARY STORAGE DEVICES

Auxiliary storage is also known as auxiliary memory or secondary storage, is the memory that supplements the main storage. This is a long term non-volatile memory. The term non-volatile memory means it stores and retains the programs and data even after the computer switched off. Unlike RAM which loses the contents when the computer is turned off and ROM, to which is not possible to add anything new, auxiliary storage devices allows the computer to record information semi-permanently, so it can be read later by the same computer or by another computer. Auxiliary storage devices are also useful in transferring data or programs from computer to another. They also function as the back-up devices which allow to back-up the valuable information that you are working on. So even if by some accident your computer crashes and the data in it is unrecoverable, we can restore it from our back-ups. The most common types of auxiliary storage devices are magnetic tapes, magnetic disks, floppy disks, hard disks etc.,



There are two types of auxiliary storage devices. This classification is based on the type of data access: sequential and random. Based on the type of access they are called sequential-access

media and random-access media. In the case of sequential-access media, the data stored in the media can only be read in sequence and to get to a particular point in the media we have to go through all the preceding points.

Magnetic tapes are examples of sequential-access media. In contrast disks are random-access memory also called as direct access media because a disk drive can access any point at random without passing through intervening points. Other examples of direct-access media are magnetic disks, optical disks, zip disks, etc

MAGNETIC TAPE:

Magnetic tape is a magnetically coated strip of plastic on which data can be encoded. Tapes for computers are similar to the tapes used to store music. Some personal computers, in fact, enable you to use normal cassette tapes. Storing data on is tapes considerably cheaper than storing data on disks. Tapes also have large storage capacities, ranging from a few hundred kilobytes to several gigabytes. Accessing data on tapes, however, is much slower than accessing data on disks.



Tapes are sequential-access media, which means that to get to a particular point on the tape, the tape must go through all the preceding points. In contrast disks are random-access media is because a disk drive can access any point at random without passing through intervening points because tapes are so slow, they are generally used only for long-term storage and backup. Data to be used regularly is almost always kept on a disk. Tapes are also used for transporting large amounts of data. Tapes come in a variety of sizes and formats. Tapes are sometimes called streamers or streaming tapes.

Table 7.1 Types of Tapes

Type	Capacity	Description
Half-inch	60MB-400MB	Half-inch tapes come both as 9 track reels and as cartridges. These tapes are relatively cheap, but require expensive tape drives.
Quarter-inch	40MB-5GB	Quarter-inch cartridges (QIC tape's) relatively inexpensive and support fast data transfer rates. QIC mini cartridges are even less expensive, but their data capacities transfer rates are smaller and their transfer rates are slower.

8-mm Helical	1GB-5GB	8-mm helical-scan cartridges use the same technology as VCR tapes and have the greatest capacity. But they require expensive tape drives and have relatively slow data transfer rates.
4-mm DAT	2GB-24GB	DAT (Digital Audio Tape) cartridges have the greatest capacity but they require expensive tape drives and have relatively slow data transfer rates.

Helical-scan Cartridge

A type of magnetic tape that uses the same technology as VCR-tapes. The term helical scan usually refers to 8-mm tapes, although 4-mm tapes called DAT tapes use the same technology. The 8-mm helical-scan tapes have data capacities from 2.5GB to 5GB.

DAT Cartridge:



DAT (Digital Audio Tape) is a type of magnetic tape that uses an ingenious scheme called helical scan to record data. A DAT cartridge is slightly larger than a credit card and contains a magnetic tape that can hold from 2 to 24 gigabytes of data. It can support data transfer rates of about 2 MBPS (Million bytes per second). Like other types of tapes, DATs are sequential-access media. The most common format for DAT cartridges is DDS (digital data storage) which is the industry standard for digital audio tape (DAT) formats. The latest format, DDS-3, specifies tapes that can hold 24 GB (the equivalent of over 40 CD PROMS) and support data transfer rates of 2 MBps.

WINCHESTER DISK

The term Winchester comes from an early type of disk drive developed by IBM that stored 30MB and has a 30-millisecond access time: so its inventors named it a winchester in honor of the 30-caliber rifle of the same name. Although modern disk drives are faster and hold more data, the basic technology is the same, so Winchester has become synonymous with hard disk.

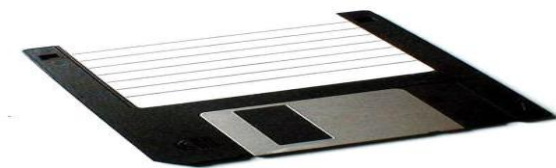
HARD DISK



Hard disk is a magnetic disk on which you can store computer data. The term hard is used to distinguish it from a soft, or floppy, disk. Hard disks hold more data and are faster than floppy disks. A hard disk, for example, can store anywhere from 10 megabytes to several gigabytes, whereas most floppies have a maximum storage capacity of 1.4 megabytes. A single hard disk usually consists several platters. Each platter requires two read/ write head one for each side. All the read/write heads are attached to a single access arm so that they cannot move independently. Each platter has the same number of tracks, and a track location that cuts across all platters is called a cylinder. For example, a typical 84 megabyte hard disk for a PC might have two platters (four sides) and 1,053 cylinders. In general, hard disks are less portable than floppies, although it is possible to buy removable hard disks. There are two types of removable hard disks: disk packs and removable cartridges.

FLOPPY DISK

Floppy disk is a soft magnetic-disk. It is called floppy because it flops if you wave it at least, the 5 1/4-inch variety does. Unlike most hard disks, floppy disks (often called floppies or diskettes) are portable, because you can remove them from a disk drive. Disk drives for floppy disks are called floppy drives. Floppy disks are slower to access than hard disks and have less storage capacity, but they are less expensive and are portable.



Floppies come in two basic sizes:

5 1/4-inch- The common size for PCs made before 1987. This type of floppy is generally capable of storing between 100K and 1.2MB (megabytes) of data. The most common sizes: are- 360K and 1.2MB.

3 1/2-inch- Floppy is something of a misnomer for these disks, as they are encased in a rigid envelope. Despite their small size, microfloppies have a larger storage capacity than their cousins—from 400K to 1.4MB of data. The most common sizes for PCs are 720K (double-density) and 1.44MB (high-density). Macintoshes support disks of 400K, 800K, and 1.2MB.

ZIP DISK

These are high-capacity floppy disk drives developed by the Iomega Corporation. Zip disks are slightly larger than the conventional floppy disks, and are about twice as thick.



ZIP DISK DRIVE

They can hold 100MB of data. Because they're relatively inexpensive and durable, they have become a popular media for backing up hard disks and for transporting large files

JAZ DISK

These are removable disk drives developed by the Iomega Corporation. The Jaz drive has a 12-ms average seek time and a transfer rate of 5.5Mbps. The removable cartridges hold 1GB of data. The fast data rates and large storage capacity make it a viable alternatives for backup storage as well as everyday use.

SUPERDISK

This a new disk storage technology, developed by the Imation Corporation that supports very high-density diskettes. Super Disk diskettes are etched with a servo pattern at the factory. This pattern is then read by the Super Disk drive to precisely align the read/ write head. The result is that a Super Disk diskette can have 2,490 tracks, as opposed to the 135 tracks that conventional 3.5-inch 1.44MB diskettes use. This higher density translates into 120 MB capacity per diskette.

Unlike the other removable disk storage solutions, such as the Zip drive, Super Disk is backward compatible with older diskettes. This means that you can use the same Super Disk drive to read and write to older 1.44MB diskettes as well as the new 120 MB Super Disk diskettes. Imation's current Super Disk drive is called the LS-120.

OPTICAL DISK

Optical Disks are a storage medium from which data is read and to which it is written by lasers. Optical disks can store much more data up to 6gigabytes (6 billion bytes)-than magnetic media, such as floppies and hard disks. There are three basic types of optical disks:

CD-ROM: Like audio CDs, CD-ROM come with data already encoded onto them. The data is permanent and can be read any number of times, but CD-ROMs cannot be modified.

WORM-This term stands for write-once ,read-many. With a WORM disk drive, you can write data onto a WORM disk, but only once. After that, the WORM disk behaves just like a CD-ROM.

Erasable-Optical disks that can be erased and loaded with new data, just like magnetic disks. These are often referred to as EO(erasable optical) disks. These three technologies are not compatible with one another; each requires a different type of disk drive and disk. Even within one category, there are many competing formats, although CD-ROMs are relatively standardized.

CD-ROM: CD-ROM, which is pronounced as ' see-dee-rom ', is the abbreviation of Compact Disc Read-Only Memory. CD-ROM is a type of optical disk capable of storing large amounts of data-up to 1GB, although the most common size is 630MB (megabytes). A single CD-ROM has the storage capacity of_700 floppy disks, enough memory to store about 300,000 text pages.

CD-ROMs are recorded by the vendor, and once recorded; they cannot be erased and filled with new data. To read a CD, you need a CD-ROM player. Also called a CD-ROM drive CD-ROM drive, a CD-ROM player is a device that can read information from a CD-ROM. CD-ROM players can be either internal, in which case they fit in a bay, or external, in which case, they generally connect to the computer's parallel port.



Parallel CD ROM players are easier to install, but they have several disadvantages: They are somewhat more expensive than internal players they use up the parallel port which means that you can't use that port for another device such a printer, and the parallel port itself may not be fast enough to handle all the data pouring through it.

There are a number of features that distinguish CD-ROM players, the most important of which is probably their speed. CD-ROM players are generally classified as single speed or some multiple of single-speed. For example, a 4x player access data at four times the speed of a single-speed player. Within these groups, however, there is some variation. Also, you need to be aware of whether the CD-ROM uses the CLV (Constant Linear Velocity) or CAV (Constant Angular Velocity) technology. The reported speeds of players that use CAV are generally not accurate because they refer only to the access speed for outer tracks. Inner tracks are accessed more slowly.

Two more precise measurements are the drives seek time and data transfer rate. The seek time, also called the access time, measures how long, on average, it takes the drive to access a particular piece of information. The data transfer rate measures how much data can be read and sent to the computer in a second.

Aside from its speed, another important feature of a CD-ROM player is its compatibility with existing standards. If you plan to run CD-ROMs in a Windows environment, you need a player that conforms to the MPC III standard. If you want to be able to view photographs stored on CD-ROM, make sure your player conforms to Kodak's Photo CD format.

Finally, you should consider how the player connects to your computer. Most CD-ROMs connect via a SCSI bus. If your computer doesn't already contain such an interface, you will need to install one. Other CD-ROMs connect to an IDE or Enhanced IDE interface, which is the one used by the hard disk drive; still others use a proprietary interface.

Almost all CD-ROMS conform to a standard size and format, so it is usually possible to load any type of CD into any ROM player. In addition, most CD-ROM players are capable of playing audio CDs, which share the same technology.

CD-ROMs are particularly well suited to information that requires large storage capacity. This includes color graphics, sound, and especially video. In recent years, as the prices of CD-ROM players have decreased, and the tools for creating new CD-ROM titles have improved, the CD-ROM industry has been expanding rapidly. To date, the most popular CD-ROM titles have been computer games and multimedia reference works.

CD-R Drive: CD-R drive, which is short for Compact Disk-Recordable drive, is a type of disk drive that can create CD-ROMs and audio CDs. This allows the users to "master" a CD-ROM or audio CD for publishing. Until recently, CD-R drives were quite expensive, but prices have dropped dramatically.

A feature of many CD-R drives, called multisession recording, enables you to keep adding data to a CD-ROM over time. This is extremely important if you want to use the CD-R drive to create backup CD-ROMs.

To create CD-ROMs and audio CDs, you'll need not only a CD-R drive, but also a CD-R software package.

Often, it is the software package, not the drive itself that determines how easy or difficult it is to create CD-ROMs. CD-R drives can also read CD-ROMs and play audio.

D-RW Disks: D-RW disk is short for CD-Rewritable disk and this is a new type of CD disk that able you to write onto it in multiple sessions. One of the problems with CD-R disks is that you can only write to them once. With CD-RW drives and disks you can treat the optical disk just like a floppy or hard disk, writing data onto it multiple times.

The first CD-RW drives became available in mid-1997. They can read CD-ROMs and can write onto today's CD-R disks, but they cannot write on CD-ROMs. Many experts believe that they'll be a popular storage medium.

MAGNETO-OPTICAL (MO) DRIVES: This is a type of disk drive that combines magnetic disk technologies with CD-ROM technologies. Like magnetic disks, MO disks can be read and written to. And like floppy disks, they are also removable. However, their storage capacity can be more than 200 megabytes, much greater than magnetic floppies. In terms of data access speed, they are faster than floppies and CD-ROMs, but not as fast as hard disk drives.

COMPUTER - HARDWARE

Hardware represents the physical and tangible components of a computer, i.e. the components that can be seen and touched.

Examples of Hardware are the following –

- **Input devices** – keyboard, mouse, etc.
- **Output devices** – printer, monitor, etc.
- **Secondary storage devices** – Hard disk, CD, DVD, etc.
- **Internal components** – CPU, motherboard, RAM, etc.

Relationship between Hardware and Software

- Hardware and software are mutually dependent on each other. Both of them must work together to make a computer produce a useful output.
- Software cannot be utilized without supporting hardware.
- Hardware without a set of programs to operate upon cannot be utilized and is useless.
- To get a particular job done on the computer, relevant software should be loaded into the hardware.

- Hardware is a one-time expense.
- Software development is very expensive and is a continuing expense.
- Different software applications can be loaded on a hardware to run different jobs.
- A software acts as an interface between the user and the hardware.
- If the hardware is the 'heart' of a computer system, then the software is its 'soul'. Both are complementary to each other.

COMPUTER - SOFTWARE

Software is a set of programs, which is designed to perform a well-defined function. A program is a sequence of instructions written to solve a particular problem.

There are two types of software –

- System Software
- Application Software

System Software

The system software is a collection of programs designed to operate, control, and extend the processing capabilities of the computer itself. System software is generally prepared by the computer manufacturers. These software products comprise of programs written in low-level languages, which interact with the hardware at a very basic level. System software serves as the interface between the hardware and the end users.

Some examples of system software are Operating System, Compilers, Interpreter, Assemblers, etc.

Here is a list of some of the most prominent features of a system software –

- Close to the system
- Fast in speed
- Difficult to design
- Difficult to understand
- Less interactive
- Smaller in size
- Difficult to manipulate
- Generally written in low-level language

Application Software

Application software products are designed to satisfy a particular need of a particular environment. All software applications prepared in the computer lab can come under the category of Application software.

Application software may consist of a single program, such as Microsoft's notepad for writing and editing a simple text. It may also consist of a collection of programs, often called a software package, which work together to accomplish a task, such as a spreadsheet package.

Examples of Application software are the following –

- Payroll Software
- Student Record Software
- Inventory Management Software
- Income Tax Software
- Railways Reservation Software
- Microsoft Office Suite Software
- Microsoft Word
- Microsoft Excel
- Microsoft PowerPoint

Features of application software are as follows –

- Close to the user
- Easy to design
- More interactive
- Slow in speed
- Generally written in high-level language
- Easy to understand
- Easy to manipulate and use
- Bigger in size and requires large storage space

Computer - Number System

When we type some letters or words, the computer translates them in numbers as computers can understand only numbers. A computer can understand the positional number system where there are only a few symbols called digits and these symbols represent different values depending on the position they occupy in the number.

The value of each digit in a number can be determined using –

- The digit
- The position of the digit in the number
- The base of the number system (where the base is defined as the total number of digits available in the number system)

Decimal Number System

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

Each position represents a specific power of the base (10). For example, the decimal number 1234 consists of the digit 4 in the units position, 3 in the tens position, 2 in the hundreds position, and 1 in the thousands position. Its value can be written as

$$(1 \times 1000) + (2 \times 100) + (3 \times 10) + (4 \times 1)$$

$$(1 \times 10^3) + (2 \times 10^2) + (3 \times 10^1) + (4 \times 10^0)$$

$$1000 + 200 + 30 + 4$$

$$1234$$

S.No.	Number System and Description
1	Binary Number System Base 2. Digits used : 0, 1
2	Octal Number System Base 8. Digits used : 0 to 7
3	Hexa Decimal Number System Base 16. Digits used: 0 to 9, Letters used : A- F

Binary Number System

Characteristics of the binary number system are as follows –

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

Example

Binary Number: 10101_2

Calculating Decimal Equivalent –

Step	Binary Number	Decimal Number
Step 1	10101 ₂	$((1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0))_{10}$
Step 2	10101 ₂	$(16 + 0 + 4 + 0 + 1)_{10}$
Step 3	10101 ₂	21 ₁₀

Note – 10101₂ is normally written as 10101.

Octal Number System

Characteristics of the octal number system are as follows –

- Uses eight digits, 0,1,2,3,4,5,6,7
- Also called as base 8 number system
- Each position in an octal number represents a **0** power of the base (8). Example 8⁰
- Last position in an octal number represents a **x** power of the base (8). Example 8^x where **x** represents the last position - 1

Example

Octal Number: 12570₈

Calculating Decimal Equivalent –

Step	Octal Number	Decimal Number
Step 1	12570 ₈	$((1 \times 8^4) + (2 \times 8^3) + (5 \times 8^2) + (7 \times 8^1) + (0 \times 8^0))_{10}$
Step 2	12570 ₈	$(4096 + 1024 + 320 + 56 + 0)_{10}$
Step 3	12570 ₈	5496 ₁₀

Hexadecimal Number System

Characteristics of hexadecimal number system are as follows –

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

Example

Hexadecimal Number: $19FDE_{16}$

Calculating Decimal Equivalent –

Step	Binary Number	Decimal Number
Step 1	$19FDE_{16}$	$((1 \times 16^4) + (9 \times 16^3) + (F \times 16^2) + (D \times 16^1) + (E \times 16^0))_{10}$
Step 2	$19FDE_{16}$	$((1 \times 16^4) + (9 \times 16^3) + (15 \times 16^2) + (13 \times 16^1) + (14 \times 16^0))_{10}$
Step 3	$19FDE_{16}$	$(65536 + 36864 + 3840 + 208 + 14)_{10}$
Step 4	$19FDE_{16}$	106462_{10}

Note – $19FDE_{16}$ is normally written as 19FDE.

Computer - Number Conversion

There are many methods or techniques which can be used to convert numbers from one base to another. In this chapter, we'll demonstrate the following –

- Decimal to Other Base System
- Other Base System to Decimal
- Other Base System to Non-Decimal
- Shortcut method - Binary to Octal
- Shortcut method - Octal to Binary
- Shortcut method - Binary to Hexadecimal
- Shortcut method - Hexadecimal to Binary

Decimal to Other Base System

Step 1 – Divide the decimal number to be converted by the value of the new base.

Step 2 – Get the remainder from Step 1 as the rightmost digit (least significant digit) of the new base number.

Step 3 – Divide the quotient of the previous divide by the new base.

Step 4 – Record the remainder from Step 3 as the next digit (to the left) of the new base number.

Repeat Steps 3 and 4, getting remainders from right to left, until the quotient becomes zero in Step 3.

The last remainder thus obtained will be the Most Significant Digit (MSD) of the new base number.

Example

Decimal Number: 29_{10}

Calculating Binary Equivalent –

Step	Operation	Result	Remainder
Step 1	$29 / 2$	14	1
Step 2	$14 / 2$	7	0
Step 3	$7 / 2$	3	1
Step 4	$3 / 2$	1	1
Step 5	$1 / 2$	0	1

As mentioned in Steps 2 and 4, the remainders have to be arranged in the reverse order so that the first remainder becomes the Least Significant Digit (LSD) and the last remainder becomes the Most Significant Digit (MSD).

Decimal Number : 29_{10} = Binary Number : 11101_2 .

Other Base System to Decimal System

Step 1 – Determine the column (positional) value of each digit (this depends on the position of the digit and the base of the number system).

Step 2 – Multiply the obtained column values (in Step 1) by the digits in the corresponding columns.

Step 3 – Sum the products calculated in Step 2. The total is the equivalent value in decimal.

Example

Binary Number: 11101_2

Calculating Decimal Equivalent –

Step	Binary Number	Decimal Number
Step 1	11101_2	$((1 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0))_{10}$
Step 2	11101_2	$(16 + 8 + 4 + 0 + 1)_{10}$
Step 3	11101_2	29_{10}

Binary Number : 11101_2 = Decimal Number : 29_{10}

Other Base System to Non-Decimal System

Step 1 – Convert the original number to a decimal number (base 10).

Step 2 – Convert the decimal number so obtained to the new base number.

Example

Octal Number : 25_8

Calculating Binary Equivalent –

Step 1 - Convert to Decimal

Step	Octal Number	Decimal Number
Step 1	25_8	$((2 \times 8^1) + (5 \times 8^0))_{10}$
Step 2	25_8	$(16 + 5)_{10}$
Step 3	25_8	21_{10}

Octal Number : $25_8 =$ Decimal Number : 21_{10}

Step 2 - Convert Decimal to Binary

Step	Operation	Result	Remainder
Step 1	$21 / 2$	10	1
Step 2	$10 / 2$	5	0
Step 3	$5 / 2$	2	1
Step 4	$2 / 2$	1	0
Step 5	$1 / 2$	0	1

Decimal Number : $21_{10} =$ Binary Number : 10101_2

Octal Number : $25_8 =$ Binary Number : 10101_2

Shortcut Method – Binary to Octal

Step 1 – Divide the binary digits into groups of three (starting from the right).

Step 2 – Convert each group of three binary digits to one octal digit.

Example

Binary Number : 10101_2

Calculating Octal Equivalent –

Step	Binary Number	Octal Number
Step 1	10101_2	010 101
Step 2	10101_2	$2_8 5_8$
Step 3	10101_2	25_8

Binary Number : $10101_2 =$ Octal Number : 25_8

Shortcut Method – Octal to Binary

Step 1 – Convert each octal digit to a 3-digit binary number (the octal digits may be treated as decimal for this conversion).

Step 2 – Combine all the resulting binary groups (of 3 digits each) into a single binary number.

Example

Octal Number : 25_8

Calculating Binary Equivalent –

Step	Octal Number	Binary Number
Step 1	25_8	$2_{10} 5_{10}$
Step 2	25_8	$010_2 101_2$
Step 3	25_8	010101_2

Octal Number : $25_8 =$ Binary Number : 10101_2

Shortcut Method – Binary to Hexadecimal

Step 1 – Divide the binary digits into groups of four (starting from the right).

Step 2 – Convert each group of four binary digits to one hexadecimal symbol.

Example: Binary Number : 10101_2

Calculating hexadecimal Equivalent –

Step	Binary Number	Hexadecimal Number
Step 1	10101_2	0001 0101
Step 2	10101_2	$1_{10} 5_{10}$
Step 3	10101_2	15_{16}

Binary Number : $10101_2 =$ Hexadecimal Number : 15_{16}

Shortcut Method - Hexadecimal to Binary

Step 1 – Convert each hexadecimal digit to a 4-digit binary number (the hexadecimal digits may be treated as decimal for this conversion).

Step 2 – Combine all the resulting binary groups (of 4 digits each) into a single binary number.

Example: Hexadecimal Number : 15_{16}

Calculating Binary Equivalent –

Step	Hexadecimal Number	Binary Number
Step 1	15_{16}	$1_{10} 5_{10}$
Step 2	15_{16}	$0001_2 0101_2$
Step 3	15_{16}	00010101_2

Hexadecimal Number : $15_{16} =$ Binary Number : 10101_2

ALGORITHM

Definition of Algorithm

- An algorithm is a finite step by step procedure, which defines a set of instructions to be executed in certain order to get the desired output.
- An algorithm is any well defined computational procedure that takes some values or set of values as input and produces some value or set of values as output.

Characteristics of Algorithm

An Algorithm must have the following characteristics.

- Input
- Output
- Definiteness or unambiguous
- Finiteness
- Effectiveness

INPUT

An algorithm should have 0 or more well defined input.

OUTPUT

An algorithm should have 1 or more well defined outputs and should match the desired results.

DEFINITENESS/ UNAMBIGUOUS

Algorithm should be clear and unambiguous. Each of its steps(or phases), and their inputs/outputs should be clear.

FINITENESS

Algorithm must terminate after a finite number of steps.

EFFECTIVENESS

It is measured in terms of time and spac.

Study of Algorithm

1. How to devise an algorithm?
2. How to validate an algorithm?
3. How to an analyse an algorithm?
4. How to test a program?

How to devise an algorithm?

Creating an algorithm is an art, which may never be fully automated. Various design techniques are used to yield good algorithms.

How to validate an algorithm?

Once an algorithm is devised, it is necessary to show that it computes the correct answer for all possible legal inputs. This process is referred as algorithm validation.

How to analyse an algorithm?

As an algorithm is executed, it uses the computers central processing unit(CPU) to perform operations and memory. Analysis of an algorithm refers to the task of determining how much computation time and storage it operation.

How to test a program?

Testing of program consists of two phases. They are

1. Debugging
2. Profiling

Debugging

Process of executing programs on sample data sets to determine whether faulty results occur, if so correct them.

Profiling

Profiling or performance measurement is the process of executing a correct program and measuring the time and space it takes to compute the results.

Notation of an algorithm

The notation used for algorithm specification must confirm to a basic set of criteria.

1. It must be concise.
2. It must be unambiguous.
3. It must be capable of machine execution.
4. It must promote elegance in the solution.

IT MUST BE CONCISE:

We must be able to describe the solution to a problem without writing multiple pages of text.

IT MUST BE UNAMBIGUOUS:

The description of the procedure must be clear, because the algorithm will be executed by a machine, which has no 'common sense' or inbuilt knowledge of the nature of the problem.

IT MUST BE CAPABLE OF MACHINE EXECUTION: The actions described must be capable of translation into precise, machine-executable operations.

IT MUST BE ELEGANCE IN THE SOLUTION:

The tool must prevent the programmer from using practises at the stage, which lead to poor programming style during implementation.

Example of an Algorithm:

Suppose, we have to develop an algorithm to convert an integer numerical score(0 t 100) scored by a student in a particular text into letter grades(A,B,C,D,E,F,O) using the following procedure.

Letter Grade	Numerical Score
'O'	100 to 90
'E'	89 to 80
'A'	79 to 70
'B'	69 to 60
'C'	59 to 50
'D'	49 to 35
'F'	Below 35

Algorithm is defined as follows:

Step 1: INPUT the score of a student.

Step 2: If the score is less than 35, then print "F": END.

Step 3: If the score is greater than or equal to 35 and less than 50, then print "D": END.

Step 4: If the score is greater than or equal to 50 and less than 60, then print "C": END.

Step 5: If the score is greater than or equal to 60 and less than 70, then print "B": END.

Step 6: If the score is greater than or equal to 70 and less than 80, then print "A":END.

Step 7: If the score is greater than or equal to 80 and less than 90, then print "E": END.







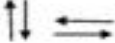
Step 8: If the score is greater than 89 then print "O": END.

Step 9: END.

FLOWCHART

- Flowchart is a graphical representation of an algorithm.
- It make use of the basic operation in programming.
- All symbols are connected among themselves to indicate the flow of information and processing.
- It is a quality improvement tool for specifying the process of the program.
- It tends to provide people with a common language or reference point when dealing with development of a program.

FLOWCHART SYMBOLS AND ITS FUCTION

Symbol	Name	Function
	Process	Indicates any type of internal operation inside the Processor or Memory
	input/output	Used for any Input / Output (I/O) operation. Indicates that the computer is to obtain data or output results
	Decision	Used to ask a question that can be answered in a binary format (Yes/No, True/False)
	Connector	Allows the flowchart to be drawn without intersecting lines or without a reverse flow.
	Predefined Process	Used to invoke a subroutine or an Interrupt program.
	Terminal	Indicates the starting or ending of the program, process, or interrupt program
	Flow Lines	Shows direction of flow.

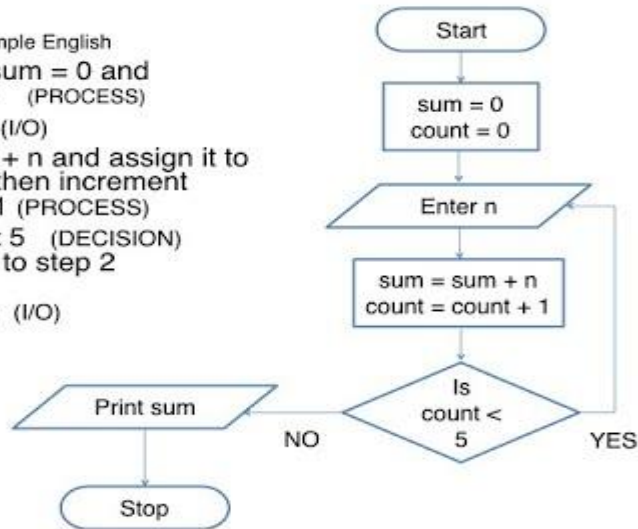
ALGORITHM AND ITS FLOWCHART

Find the sum of 5 numbers

Flowchart

Algorithm in simple English

1. Initialize $sum = 0$ and $count = 0$ (PROCESS)
2. Enter n (I/O)
3. Find $sum + n$ and assign it to sum and then increment $count$ by 1 (PROCESS)
4. Is $count < 5$ (DECISION)
if YES go to step 2
else
Print sum (I/O)



PROGRAMMING IN C

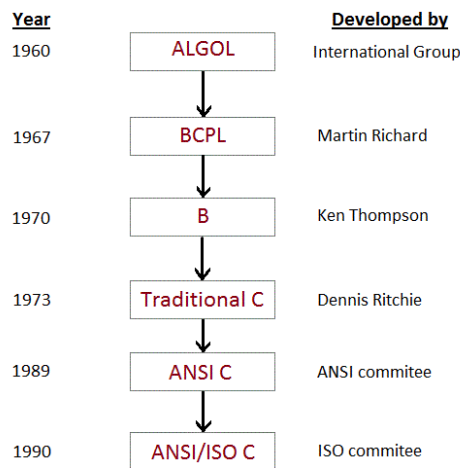
UNIT I

INTRODUCTION OF C LANGUAGE

C is a structured programming language developed by Dennis Ritchie in 1973 at Bell Laboratories. It is one of the most popular computer languages today because of its structure, high-level abstraction, machine independent feature etc. C language was developed to write the UNIX operating system, hence it is strongly associated with UNIX, which is one of the most popular network operating system in use today and heart of internet data superhighway.

History of C language

C language has evolved from three different structured language ALGOL, BCPL and B Language. It uses many concepts from these languages while introduced many new concepts such as datatypes, struct, pointer etc. In 1988, the language was formalised by **American National Standard Institute(ANSI)**. In 1990, a version of C language was approved by the **International Standard Organisation(ISO)** and that version of C is also referred to as C89.



C Program and its Structure

- Pre-processor
- Header file
- Function
- Comments

(i) PREPROCESSOR

#include is the first word of any C program. It is also known as a **pre-processor**. The task of a pre-processor is to initialize the environment of the program, i.e to link the program with the header files required. So, when we say #include <stdio.h>, it is to inform the compiler to include the **stdio.h** header file to the program before executing it.

(ii) Header file

A Header file is a collection of built-in(readymade) functions. Header files contain definitions of the functions which can be incorporated into any C program by using pre-processor #include statement with the header file. To use any of the standard functions, the appropriate header file must be included. This is done at the beginning of the C source file. To use the printf() function in a program, which is used to display anything on the screen, the line #include <stdio.h> is required because the header file **stdio.h** contains the printf() function. All header files will have an extension .h

(iii) main() function

main() function is a function that must be there in every C program. Everything inside this function in a C program will be executed. In the above example, int written before the main() function is the return type of main() function. we will discuss about it in detail later. The curly braces { } just after the main() function encloses the body of main() function.

(iv) Comments

We can add comments in our program to describe what we are doing in the program. These comments are ignored by the compiler and are not executed.

To add a single line comment, start it by adding two forward slashes // followed by the comment.

To add multiline comment, enclose it between /* ... */

(V) Return statement - return 0;

A return statement is just meant to define the end of any C program.

All the C programs can be written and edited in normal text editors like Notepad or Notepad++ and must be saved with a file name with extension as .c

FIRST C PROGRAM

```
#include <stdio.h>
int main()
{
printf("Hello C Language");
return 0;
}
```

CONSTANTS

Constants refer to fixed values that the program may not alter during its execution. These fixed values are also called **literals**.

Constants can be of any of the basic data types like *an integer constant, a floating constant, a character constant, or a string literal*. There are enumeration constants as well.

Constants are treated just like regular variables except that their values cannot be modified after their definition.

Integer Literals

An integer literal can be a decimal, octal, or hexadecimal constant. A prefix specifies the base or radix: 0x or 0X for hexadecimal, 0 for octal, and nothing for decimal.

An integer literal can also have a suffix that is a combination of U and L, for unsigned and long, respectively. The suffix can be uppercase or lowercase and can be in any order.

Floating-point Literals

A floating-point literal has an integer part, a decimal point, a fractional part, and an exponent part. You can represent floating point literals either in decimal form or exponential form.

Character Constants

Character literals are enclosed in single quotes, e.g., 'x' can be stored in a simple variable of **char** type. A character literal can be a plain character (e.g., 'x'), an escape sequence (e.g., '\t'), or a universal character (e.g., '\u02C0'). There are certain characters in C that represent special meaning when preceded by a backslash for example, newline (\n) or tab (\t).

String Literals

String literals or constants are enclosed in double quotes "". A string contains characters that are similar to character literals: plain characters, escape sequences, and universal characters.

You can break a long line into multiple lines using string literals and separating them using white spaces.

Variables

In C language we can store it in a memory space and name the memory space so that it becomes easier to access it.

The naming of an address is known as **variable**. Variable is the name of memory location. Unlike constant, variables are changeable, we can change value of a variable during execution of a program. A programmer can choose a meaningful variable name. Example : average, height, age, total etc.

Datatype of Variable

A **variable** in C language must be given a type, which defines what type of data the variable will hold.

It can be:

- char: Can hold/store a character in it.
- int: Used to hold an integer.
- float: Used to hold a float value.
- double: Used to hold a double value.
- void

Rules to name a Variable

1. Variable name must not start with a digit.
2. Variable name can consist of alphabets, digits and special symbols like underscore _.
3. Blank or spaces are not allowed in variable name.
4. Keywords are not allowed as variable name.
5. Upper and lower case names are treated as different, as C is case-sensitive, so it is suggested to keep the variable names in lower case.

Variable declaration and initialization

Syntax:

Datatype variable name;

Example: int a;

int a=10;

Data types

Data types specify how we enter data into our programs and what type of data we enter. C language has some predefined set of data types to handle various kinds of data that we can use in our program. These datatypes have different storage capacities.

C language supports 2 different type of data types:

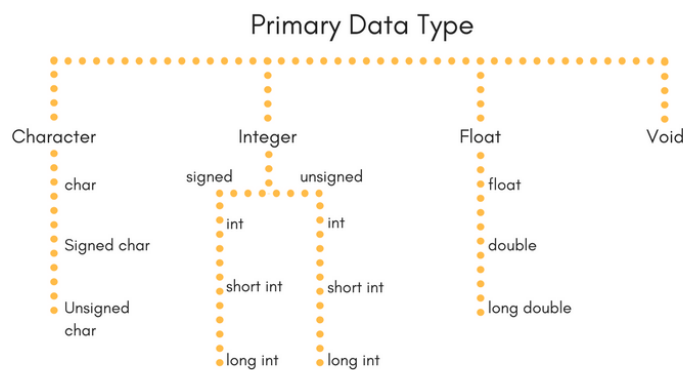
1. **Primary data types:**

These are fundamental data types in C namely integer(int), floating point(float), character(char) and void.

2. Derived data types:

Derived data types are nothing but primary datatypes but a little twisted or grouped together like **array**, **stucture**, **union** and **pointer**. These are discussed in details later.

Data type determines the type of data a variable will hold. If a variable x is declared as int. it means x can hold only integer values. Every variable which is used in the program must be declared as what data-type it is.



Integer type

Integers are used to store whole numbers.

Size and range of Integer type on 16-bit machine:

Type	Size(bytes)	Range
int or signed int	2	-32,768 to 32767
unsigned int	2	0 to 65535
short int or signed short int	1	-128 to 127
unsigned short int	1	0 to 255
long int or signed long int	4	-2,147,483,648 to 2,147,483,647
unsigned long int	4	0 to 4,294,967,295

Floating point type

Floating types are used to store real numbers.

Size and range of Integer type on 16-bit machine

Type	Size(bytes)	Range
------	-------------	-------

Float	4	3.4E-38 to 3.4E+38
double	8	1.7E-308 to 1.7E+308
long double	10	3.4E-4932 to 1.1E+4932

Character type

Character types are used to store characters value.

Size and range of Integer type on 16-bit machine

Type	Size(bytes)	Range
char or signed char	1	-128 to 127
unsigned char	1	0 to 255

void type

void type means no value. This is usually used to specify the type of functions which returns nothing. We will get acquainted to this datatype as we start learning more advanced topics in C language, like functions, pointers etc.

Operators

C language supports a rich set of built-in operators. An operator is a symbol that tells the compiler to perform a certain mathematical or logical manipulation. Operators are used in programs to manipulate data and variables.

C operators can be classified into following types:

- Arithmetic operators
- Relational operators
- Logical operators
- Bitwise operators
- Assignment operators
- Conditional operators
- Special operators

Arithmetic operators

C supports all the basic arithmetic operators. The following table shows all the basic arithmetic operators.

Operator	Description
+	adds two operands
-	subtract second operands from first
*	multiply two operand
/	divide numerator by denominator
%	remainder of division

++	Increment operator - increases integer value by one
--	Decrement operator - decreases integer value by one

Relational operators

The following table shows all relation operators supported by C.

Operator	Description
==	Check if two operand are equal
!=	Check if two operand are not equal.
>	Check if operand on the left is greater than operand on the right
<	Check operand on the left is smaller than right operand
>=	check left operand is greater than or equal to right operand
<=	Check if operand on left is smaller than or equal to right operand

Logical operators

C language supports following 3 logical operators. Suppose a = 1 and b = 0,

Operator	Description	Example
&&	Logical AND	(a && b) is false
	Logical OR	(a b) is true
!	Logical NOT	(!a) is false

Bitwise operators

Bitwise operators perform manipulations of data at **bit level**. These operators also perform **shifting of bits** from right to left. Bitwise operators are not applied to float or double(These are datatypes, we will learn about them in the next tutorial).

Operator	Description
&	Bitwise AND
	Bitwise OR
^	Bitwise exclusive OR
<<	left shift
>>	right shift

Now lets see truth table for bitwise &, | and ^

a	b	a & b	a b	a ^ b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

The bitwise **shift** operator, shifts the bit value. The left operand specifies the value to be shifted and the right operand specifies the number of positions that the bits in the value have to be shifted. Both operands have the same precedence.

Assignment Operators

Assignment operators supported by C language are as follows.

Operator	Description	Example
=	assigns values from right side operands to left side operand	a=b
+=	adds right operand to the left operand and assign the result to left	a+=b is same as a=a+b
-=	subtracts right operand from the left operand and assign the result to left operand	a-=b is same as a=a-b
=	multiply left operand with the right operand and assign the result to left operand	a=b is same as a=a*b
/=	divides left operand with the right operand and assign the result to left operand	a/=b is same as a=a/b
%=	calculate modulus using two operands and assign the result to left operand	a%=b is same as a=a%b

Conditional operator

The conditional operators in C language are known by two more names

1. **Ternary Operator**
2. **? : Operator**

It is actually the if condition that we use in C language decision making, but using conditional operator, we turn the if condition statement into a short and simple operator.

The syntax of a conditional operator is :

expression 1 ? expression 2: expression 3

Explanation:

- The question mark "?" in the syntax represents the **if** part.
- The first expression (expression 1) generally returns either true or false, based on which it is decided whether (expression 2) will be executed or (expression 3)
- If (expression 1) returns true then the expression on the left side of " : " i.e (expression 2) is executed.
- If (expression 1) returns false then the expression on the right side of " : " i.e (expression 3) is executed.

Special operator

Operator	Description	Example
sizeof	Returns the size of an variable	sizeof(x) return size of the variable x
&	Returns the address of an variable	&x ; return address of the variable x
*	Pointer to a variable	*x ; will be pointer to a variable x

Expressions

An expression is a combination of variables constants and operators written according to the syntax of C language. In C every expression evaluates to a value i.e., every expression results in some value of a certain type that can be assigned to a variable. Some examples of C expressions are shown in the table given below.

Example

a*b-c/d

Evaluation of Expressions

Expressions are evaluated using an assignment statement of the form

Variable = expression;

Variable is any valid C variable name. When the statement is encountered, the expression is evaluated first and then replaces the previous value of the variable on the left hand side. All variables used in the expression must be assigned values before evaluation is attempted.

precedence in Arithmetic Operators

An arithmetic expression without parenthesis will be evaluated from left to right using the rules of precedence of operators. There are two distinct priority levels of arithmetic operators in C.

High priority * / %

Low priority + -

Rules for evaluation of expression

- First parenthesized sub expression left to right are evaluated.
- If parenthesis are nested, the evaluation begins with the innermost sub expression.
- The precedence rule is applied in determining the order of application of operators in evaluating sub expressions.
- The associability rule is applied when two or more operators of the same precedence level appear in the sub expression.
- Arithmetic expressions are evaluated from left to right using the rules of precedence.
- When Parenthesis are used, the expressions within parenthesis assume highest priority.

Operator precedence and associativity

Each operator in C has a precedence associated with it. The precedence is used to determine how an expression involving more than one operator is evaluated. There are distinct levels of

precedence and an operator may belong to one of these levels. The operators of higher precedence are evaluated first. The operators of same precedence are evaluated from right to left or from left to right depending on the level. This is known as associativity property of an operator.

The table given below gives the precedence of each operator.

Order	Category	Operator	Operation	Associativity
1	Highest precedence	() [] ? : :	Function call	L ? R Left to Right
2	Unary	! ~ + - ++ - - & * Size of	Logical negation (NOT) Bitwise 1's complement Unary plus Unary minus Pre or post increment Pre or post decrement Address Indirection Size of operand in bytes	R ? L Right -> Left
3	Member Access	.* ?*	Dereference Dereference	L ? R
4	Multiplication	* / %	Multiply Divide Modulus	L ? R
5	Additive	+ -	Binary Plus Binary Minus	L ? R
6	Shift	<< >>	Shift Left Shift Right	L ? R
7	Relational	< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	L ? R
8	Equality	== !=	Equal to Not Equal to	L ? R
9	Bitwise AAND	&	Bitwise AND	L ? R
10	Bitwise XOR	^	Bitwise XOR	L ? R
11	Bitwise OR		Bitwise OR	L ? R
12	Logical AND	&&	Logical AND	L ? R
14	Conditional	? :	Ternary Operator	R ? L
15	Assignment	= *= %= /=	Assignment Assign product Assign remainder Assign quotient	R ? L
		+= - =	Assign sum Assign difference	

		&= ^= = <<= >>=	Assign bitwise AND Assign bitwise XOR Assign bitwise OR Assign left shift Assign right shift	
16	Comma	,	Evaluate	L ? R

UNIT – II

MANAGING INPUT AND OUTPUT OPERATIONS

C Input and Output

Input means to provide the program with some data to be used in the program and **Output** means to display data on screen or write the data to a printer or a file.

C programming language provides many built-in functions to read any given input and to display data on screen when there is a need to output the result.

scanf() and printf() functions

The standard input-output header file, named stdio.h contains the definition of the functions printf() and scanf(), which are used to display output on screen and to take input from user respectively. scanf() or printf() functions. It is known as **format string** and this informs the scanf() function, what type of input to expect and in printf() it is used to give a heads up to the compiler, what type of output to expect.

Format String	Meaning
%d	Scan or print an integer as signed decimal number
%f	Scan or print a floating point number
%c	To scan or print a character
%s	To scan or print a character string. The scanning ends at whitespace.

SYNTAX:

```
printf("studytonight");
```

SYNTAX:

```
scanf("control string", & variable name);
```

getchar() & putchar() functions

This function reads only single character at a time. You can use this method in a [loop](#) in case you want to read more than one character. The putchar() function displays the character passed to it

on the screen and returns the same character. This function too displays only a single character at a time.

```
c = getchar();  
putchar(c);
```

gets() & puts() functions

The gets() function reads a line from **stdin**(standard input) into the buffer pointed to by str [pointer](#), until either a terminating newline or EOF (end of file) occurs. The puts() function writes the string str and a trailing newline to **stdout**.

str → This is the pointer to an array of chars where the C string is stored.

Decision making statements

Decision making is about deciding the order of execution of statements based on certain conditions or repeat a group of statements until certain specified conditions are met. C language handles decision-making by supporting the following statements,

- if statement
- switch statement
- conditional operator statement (? : operator)
- goto statement

Decision making with if statement

The if statement may be implemented in different forms depending on the complexity of conditions to be tested. The different forms are,

1. Simple if statement
2. if...else statement
3. Nested if...else statement
4. Using else if statement

Simple if statement

The general form of a simple if statement is,

```
if(expression)  
{  
    statement inside;  
}  
statement outside;
```

If the *expression* returns true, then the **statement-inside** will be executed, otherwise **statement-inside** is skipped and only the **statement-outside** is executed.

Example:

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

```
void main( )  
{  
    int x, y;  
    x = 15;  
    y = 13;  
    if (x > y )  
    {  
        printf("x is greater than y");  
    }  
}
```

x is greater than y

if...else statement

The general form of a simple if...else statement is,

```
if(expression)  
{  
    statement block1;  
}  
else  
{  
    statement block2;  
}
```

If the *expression* is true, the **statement-block1** is executed, else **statement-block1** is skipped and **statement-block2** is executed.

Example:

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

```
void main( )  
{  
    int x, y;  
    x = 15;  
    y = 18;  
    if (x > y )  
    {  
        printf("x is greater than y");  
    }  
}
```



```

    }
    else
    {
        printf("y is greater than x");
    }
}

```

y is greater than x

Nested if...else statement

The general form of a nested if...else statement is,

```

if( expression )
{
    if( expression1 )
    {
        statement block1;
    }
    else
    {
        statement block2;
    }
}
else
{
    statement block3;
}

```

if *expression* is false then **statement-block3** will be executed, otherwise the execution continues and enters inside the first if to perform the check for the next if block, where if *expression 1* is true the **statement-block1** is executed otherwise **statement-block2** is executed.

Example:

```

#include <stdio.h>

void main( )
{
    int a, b, c;
    printf("Enter 3 numbers...");
    scanf("%d%d%d",&a, &b, &c);
    if(a > b)
    {
        if(a > c)
        {
            printf("a is the greatest");
        }
    }
}

```

```

    else
    {
        printf("c is the greatest");
    }
}
else
{
    if(b > c)
    {
        printf("b is the greatest");
    }
    else
    {
        printf("c is the greatest");
    }
}
}

```

else if ladder

The general form of else-if ladder is,

```

if(expression1)
{
    statement block1;
}
else if(expression2)
{
    statement block2;
}
else if(expression3 )
{
    statement block3;
}
else
    default statement;

```

The expression is tested from the top(of the ladder) downwards. As soon as a **true** condition is found, the statement associated with it is executed.

Example :

```

#include <stdio.h>

void main( )
{

```

```

int a;
printf("Enter a number...");
scanf("%d", &a);
if(a%5 == 0 && a%8 == 0)
{
    printf("Divisible by both 5 and 8");
}
else if(a%8 == 0)
{
    printf("Divisible by 8");
}
else if(a%5 == 0)
{
    printf("Divisible by 5");
}
else
{
    printf("Divisible by none");
}
}

```

Switch statement

switch statement is a control statement that allows us to choose only one choice among the many given choices. The expression in switch evaluates to return an integral value, which is then compared to the values present in different cases. It executes that block of code which matches the case value. If there is no match, then **default** block is executed(if present). The general form of switch statement is,

```

switch(expression)
{
    case value-1:
        block-1;
        break;
    case value-2:
        block-2;
        break;
    case value-3:
        block-3;
        break;
    case value-4:
        block-4;
        break;
    default:
        default-block;
        break;
}

```

Rules for using switch statement

1. The expression (after switch keyword) must yield an **integer** value i.e the expression should be an integer or a variable or an expression that evaluates to an integer.
2. The case **label** values must be unique.
3. The case label must end with a colon(:)
4. The next line, after the **case** statement, can be any valid C statement.

Looping statements

loops are used to execute a set of statements repeatedly until a particular condition is satisfied.

Types of Loop

There are 3 types of Loop in C language, namely:

1. while loop
2. for loop
3. do while loop

while loop

while loop can be addressed as an **entry control** loop. It is completed in 3 steps.

- Variable initialization.(e.g int x = 0;)
- condition(e.g while(x <= 10))
- Variable increment or decrement (x++ or x-- or x = x + 2)

Syntax :

```
variable initialization;
while(condition)
{
    statements;
    variable increment or decrement;
}
```

for loop

for loop is used to execute a set of statements repeatedly until a particular condition is satisfied. We can say it is an **open ended loop**.. General format is,

```
for(initialization; condition; increment/decrement)
{
    statement-block;
}
```

In for loop we have exactly two semicolons, one after initialization and second after the condition. In this loop we can have more than one initialization or increment/decrement, separated using comma operator. But it can have only one **condition**.

The for loop is executed as follows:

1. It first evaluates the initialization code.
2. Then it checks the condition expression.
3. If it is **true**, it executes the for-loop body.
4. Then it evaluate the increment/decrement condition and again follows from step 2.
5. When the condition expression becomes **false**, it exits the loop.

Example: Program to print first 10 natural numbers

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

```
void main( )  
{  
    int x;  
    for(x = 1; x <= 10; x++)  
    {  
        printf("%d\t", x);  
    }  
}
```

```
1 2 3 4 5 6 7 8 9 10
```

Nested for loop

We can also have nested for loops, i.e one for loop inside another for loop. Basic syntax is,

```
for(initialization; condition; increment/decrement)  
{  
    for(initialization; condition; increment/decrement)  
    {  
        statement ;  
    }  
}
```

do while loop

In some situations it is necessary to execute body of the loop before testing the condition. Such situations can be handled with the help of do-while loop. do statement evaluates the body of the loop first and at the end, the condition is checked using while statement. It means that the body of the loop will be executed at least once, even though the starting condition inside while is initialized to be **false**. General syntax is,

```
do
{
    ....
    ....
}
while(condition)
```

Example: Program to print first 10 multiples of 5.

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

```
void main()
{
    int a, i;
    a = 5;
    i = 1;
    do
    {
        printf("%d\t", a*i);
        i++;
    }
    while(i <= 10);
}
```

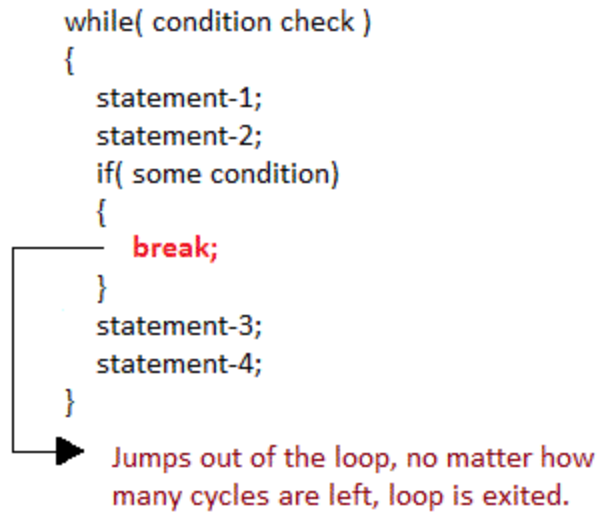
5 10 15 20 25 30 35 40 45 50

Jumping Out of Loops

Sometimes, while executing a loop, it becomes necessary to skip a part of the loop or to leave the loop as soon as certain condition becomes **true**. This is known as jumping out of loop.

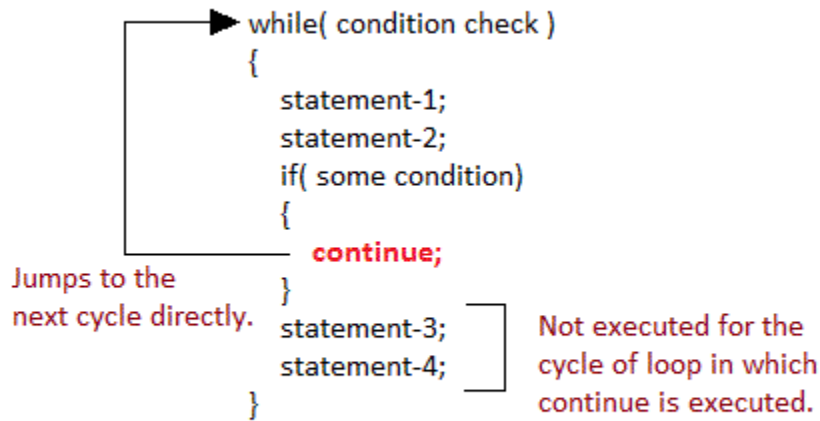
1) break statement

When break statement is encountered inside a loop, the loop is immediately exited and the program continues with the statement immediately following the loop.



2) *continue statement*

It causes the control to go directly to the test-condition and then continue the loop process. On encountering continue, cursor leave the current cycle of loop, and starts with the next cycle.



UNIT – III

ARRAYS

Arrays:

In C language, arrays are referred to as structured data types. An array is defined as **finite ordered collection of homogenous** data, stored in contiguous memory locations.

Here the words,

- **finite** *means* data range must be defined.
- **ordered** *means* data must be stored in continuous memory addresses.
- **homogenous** *means* data must be of similar data type.

Example where arrays are used,

- to store list of Employee or Student names,
- to store marks of students,
- or to store list of numbers or characters etc.

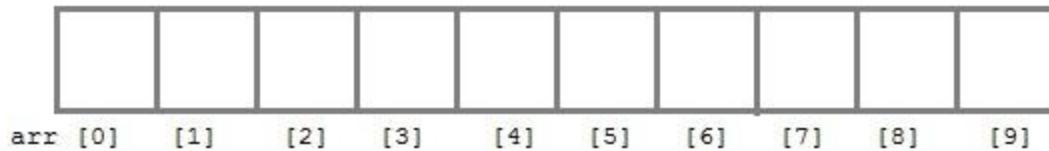
Declaring an Array

Like any other variable, arrays must be declared before they are used. General form of array declaration is,

```
data-type variable-name[size];
```

```
/* Example of array declaration */
```

```
int arr[10];
```

Here int is the data type, arr is the name of the array and 10 is the size of array. It means array arr can only contain 10 elements of int type.

Index of an array starts from 0 to **size-1** i.e first element of arr array will be stored at arr[0] address and the last element will occupy arr[9].

Initialization of an Array

After an array is declared it must be initialized. Otherwise, it will contain **garbage** value(any random value). An array can be initialized at either **compile time** or at **runtime**.

Compile time Array initialization

Compile time initialization of array elements is same as ordinary variable initialization. The general form of initialization of array is,

```
data-type array-name[size] = { list of values };
```

```
/* Here are a few examples */
```

```
int marks[4]={ 67, 87, 56, 77 }; // integer array initialization
```

```
float area[5]={ 23.4, 6.8, 5.5 }; // float array initialization
```

```
int marks[4]={ 67, 87, 56, 77, 59 }; // Compile time error
```

One important thing to remember is that when you will give more initializer(array elements) than the declared array size than the **compiler** will give an error.

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

```
void main()
```

```
{
    int i;
    int arr[] = {2, 3, 4}; // Compile time array initialization
    for(i = 0 ; i < 3 ; i++)
    {
        printf("%d\t",arr[i]);
    }
}
```

Runtime Array initialization

An array can also be initialized at runtime using scanf() function. This approach is usually used for initializing large arrays, or to initialize arrays with user specified values. Example,

```
#include<stdio.h>

void main()
{
    int arr[4];
    int i, j;
    printf("Enter array element");
    for(i = 0; i < 4; i++)
    {
        scanf("%d", &arr[i]); //Run time array initialization
    }
    for(j = 0; j < 4; j++)
    {
        printf("%d\n", arr[j]);
    }
}
```

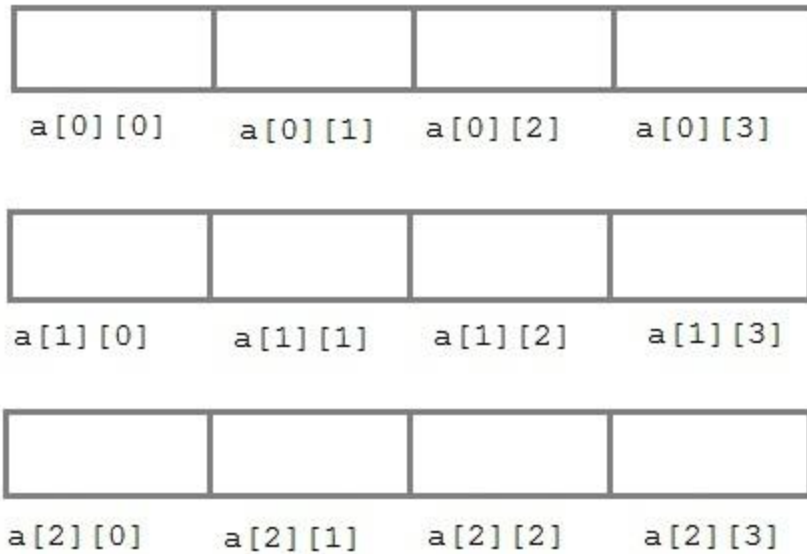
Two dimensional Arrays

C language supports multidimensional arrays also. The simplest form of a multidimensional array is the two-dimensional array. Both the row's and column's index begins from 0.

Two-dimensional arrays are declared as follows,

```
data-type array-name[row-size][column-size]
```

```
/* Example */
int a[3][4];
```



An array can also be declared and initialized together. For example,

```
int arr[][3] = {
    {0,0,0},
    {1,1,1}
};
```

Note: We have not assigned any row value to our array in the above example. It means we can initialize any number of rows. But, we must always specify number of columns, else it will give a compile time error. Here, a 2*3 multi-dimensional matrix is created.

Runtime initialization of a two dimensional Array

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

```
void main()
{
    int arr[3][4];
    int i, j, k;
    printf("Enter array element");
    for(i = 0; i < 3;i++)
    {
        for(j = 0; j < 4; j++)
        {
            scanf("%d", &arr[i][j]);
        }
    }
    for(i = 0; i < 3; i++)
    {
```

```

    for(j = 0; j < 4; j++)
    {
        printf("%d", arr[i][j]);
    }
}

```

String and Character Array

String is a sequence of characters that is treated as a single data item and terminated by null character '\0'. Remember that C language does not support strings as a data type. A **string** is actually one-dimensional array of characters in C language. These are often used to create meaningful and readable programs.

For example: The string "hello world" contains 12 characters including '\0' character which is automatically added by the compiler at the end of the string.

Declaring and Initializing a string variables:

There are different ways to initialize a character array variable.

```
char name[13] = "StudyTonight";    // valid character array initialization
```

```
char name[10] = {'L','e','s','s','o','n','s','\0'};    // valid initialization
```

Remember that when you initialize a character array by listing all of its characters separately then you must supply the '\0' character explicitly.

Some examples of illegal initialization of character array are,

```
char ch[3] = "hell";    // Illegal
```

```
char str[4];
str = "hell";    // Illegal
```

String Input and Output:

Input function scanf() can be used with %s format specifier to read a string input from the terminal. But there is one problem with scanf() function, it terminates its input on the first white space it encounters. Therefore if you try to read an input string "Hello World" using scanf() function, it will only read **Hello** and terminate after encountering white spaces.

However, C supports a format specification known as the **edit set conversion code %[..]** that can be used to read a line containing a variety of characters, including white spaces.

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

void main()
{
    char str[20];
    printf("Enter a string");
    scanf("%[^\n]", &str); //scanning the whole string, including the white spaces
    printf("%s", str);
}
```

Another method to read character string with white spaces from terminal is by using the gets() function.

```
char text[20];
gets(text);
printf("%s", text);
```

String Handling Functions:

C language supports a large number of string handling functions that can be used to carry out many of the string manipulations. These functions are packaged in **string.h** library. Hence, you must include **string.h** header file in your programs to use these functions.

The following are the most commonly used string handling functions.

Method	Description
strcat()	It is used to concatenate(combine) two strings
strlen()	It is used to show length of a string
strrev()	It is used to show reverse of a string
strcpy()	Copies one string into another
strcmp()	It is used to compare two string

strcat() function

```
strcat("hello", "world");
```

strcat() function will add the string **"world"** to **"hello"** i.e it will output helloworld.

strlen() function

strlen() function will return the length of the string passed to it.

```
int j;
```

```
j = strlen("studytonight");  
printf("%d",j);
```

12

strcmp() function

strcmp() function will return the ASCII difference between first unmatching character of two strings.

```
int j;  
j = strcmp("study", "tonight");  
printf("%d",j);
```

-1

strcpy() function

It copies the second string argument to the first string argument.

```
#include<stdio.h>  
#include<string.h>  
  
int main()  
{  
    char s1[50];  
    char s2[50];  
  
    strcpy(s1, "StudyTonight"); //copies "studytonight" to string s1  
    strcpy(s2, s1); //copies string s1 to string s2  
  
    printf("%s\n", s2);  
  
    return(0);  
}
```

StudyTonight

strrev() function

It is used to reverse the given string expression.

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

```
int main()
{
    char s1[50];

    printf("Enter your string: ");
    gets(s1);
    printf("\nYour reverse string is: %s",strrev(s1));
    return(0);
}
```

Enter your string: studytonight Your reverse string is: thginotyduts

Functions

A **function** is a block of code that performs a particular task.

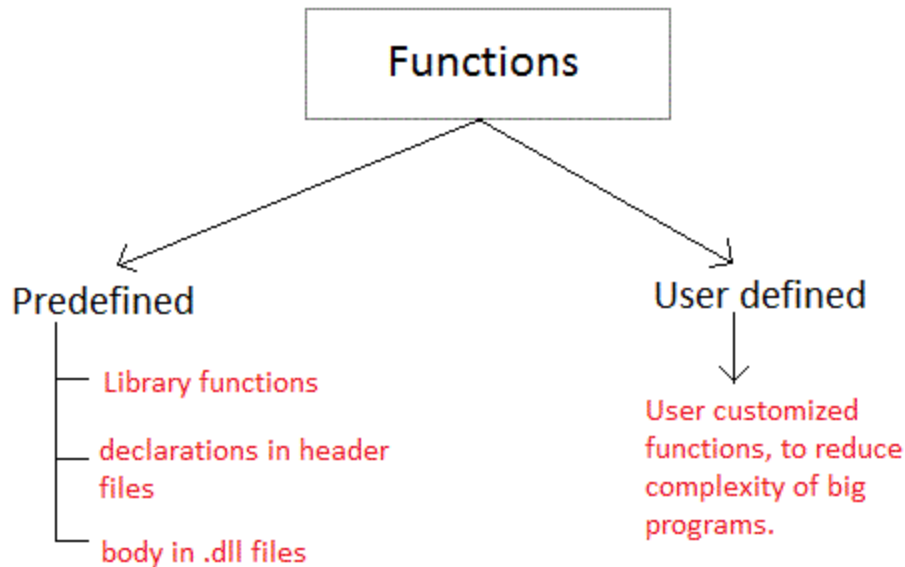
There are many situations where we might need to write same line of code for more than once in a program.

C language provides an approach in which you can declare and define a group of statements once in the form of a function and it can be called and used whenever required.

These functions defined by the user are also know as **User-defined Functions**

C functions can be classified into two categories,

1. **Library functions**
2. **User-defined functions**



Library functions are those functions which are already defined in C library, example printf(), scanf(), strcat() etc. You just need to include appropriate header files to use these functions. These are already declared and defined in C libraries.

A **User-defined functions** on the other hand, are those functions which are defined by the user at the time of writing program. These functions are made for code reusability and for saving time and space.

Benefits of Using Functions

1. It provides modularity to your program's structure.
2. It makes your code reusable. You just have to call the function by its name to use it, wherever required.
3. In case of large programs with thousands of code lines, debugging and editing becomes easier if you use functions.
4. It makes the program more readable and easy to understand.

Function Declaration

General syntax for function declaration is,

```
returntype functionName(type1 parameter1, type2 parameter2,...);
```

Like any variable or an array, a function must also be declared before its used. Function declaration informs the compiler about the function name, parameters it accepts, and its return type. The actual body of the function can be defined separately. It's also called as **Function Prototyping**. Function declaration consists of 4 parts.

- returntype

- function name
- parameter list
- terminating semicolon

returntype :

When a function is declared to perform some sort of calculation or any operation and is expected to provide with some result at the end, in such cases, a return statement is added at the end of function body. Return type specifies the type of value(int, float, char, double) that function is expected to return to the program which called the function.

Note: In case your function doesn't return any value, the return type would be void.

functionName

Function name is an [identifier](#) and it specifies the name of the function. The function name is any valid C identifier and therefore must follow the same naming rules like other variables in C language.

parameter list

The parameter list declares the type and number of arguments that the function expects when it is called. Also, the parameters in the parameter list receives the argument values when the function is called. They are often referred as **formal parameters**.

Function definition Syntax:

the general syntax of function definition is,

```
returntype functionName(type1 parameter1, type2 parameter2,...)
{
    // function body goes here
}
```

The first line *returntype* **functionName(type1 parameter1, type2 parameter2,...)** is known as **function header** and the statement(s) within curly braces is called **function body**.

functionbody

The function body contains the declarations and the statements(algorithm) necessary for performing the required task. The body is enclosed within curly braces { ... } and consists of three parts.

- **local** variable declaration(if required).
- **function statements** to perform the task inside the function.
- a **return** statement to return the result evaluated by the function(if return type is void, then no return statement is required).

Calling a function

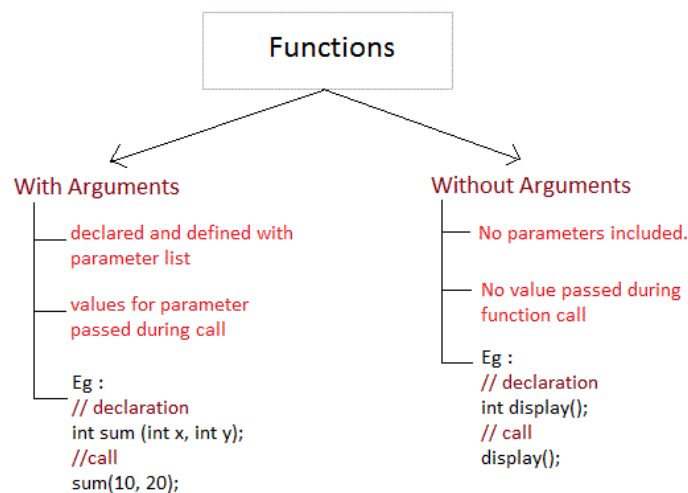
When a function is called, control of the program gets transferred to the function.

```
functionName(argument1, argument2,...);
```

In the example above, the statement `multiply(i, j);` inside the `main()` function is function call.

Passing Arguments to a function

Arguments are the values specified during the function call, for which the formal parameters are declared while defining the function.



It is possible to have a function with parameters but no return type. It is not necessary, that if a function accepts parameter(s), it must return a result too.

```

#include<stdio.h>

int multiply(int a, int b);

int main()
{
    ... ..
    result = multiply(i, j);
    ... ..
}

int multiply(int a, int b)
{
    ... ..
}

```

providing arguments while calling function

While declaring the function, we have declared two parameters a and b of type int. Therefore, while calling that function, we need to pass two arguments, else we will get compilation error. And the two arguments passed should be received in the function definition, which means that the function header in the function definition should have the two parameters to hold the argument values. These received arguments are also known as **formal parameters**. The name of the variables while declaring, calling and defining a function can be different.

Returning a value from function

A function may or may not return a result. But if it does, we must use the return statement to output the result. return statement also ends the function execution, hence it must be the last statement of any function. If you write any statement after the return statement, it won't be executed.

```

#include<stdio.h>

int multiply(int a, int b);

int main()
{
    ... ..
    result = multiply(i, j);
    ... ..
}

int multiply(int a, int b)
{
    ... ..
    return a*b;
}

```

The value returned by the function must be stored in a variable.

The datatype of the value returned using the return statement should be same as the return type mentioned at function declaration and definition. If any of it mismatches, you will get compilation error.

In the next tutorial, we will learn about the different types of user defined functions in C language and the concept of Nesting of functions which is used in recursion.

Type of User-defined Functions :

There can be 4 different types of user-defined functions, they are:

1. Function with no arguments and no return value
2. Function with no arguments and a return value
3. Function with arguments and no return value
4. Function with arguments and a return value

Below, we will discuss about all these types, along with program examples.

1.Function with no arguments and no return value

Such functions can either be used to display information or they are completely dependent on user inputs.

Below is an example of a function, which takes 2 numbers as input from user, and display which is the greater number.

```
#include<stdio.h>

void greatNum();    // function declaration

int main()
{
    greatNum();    // function call
    return 0;
}

void greatNum()    // function definition
{
    int i, j;
    printf("Enter 2 numbers that you want to compare...");
    scanf("%d%d", &i, &j);
    if(i > j) {
        printf("The greater number is: %d", i);
    }
    else {
        printf("The greater number is: %d", j);
    }
}
```

2.Function with no arguments and a return value

We have modified the above example to make the function `greatNum()` return the number which is greater amongst the 2 input numbers.

```
#include<stdio.h>

int greatNum();    // function declaration

int main()
{
    int result;
    result = greatNum();    // function call
    printf("The greater number is: %d", result);
    return 0;
}

int greatNum()    // function definition
{
    int i, j, greaterNum;
    printf("Enter 2 numbers that you want to compare...");
    scanf("%d%d", &i, &j);
    if(i > j) {
        greaterNum = i;
    }
    else {
        greaterNum = j;
    }
    // returning the result
    return greaterNum;
}
```

3.Function with arguments and no return value

We are using the same function as example again and again, to demonstrate that to solve a problem there can be many different ways.

This time, we have modified the above example to make the function `greatNum()` take two int values as arguments, but it will not be returning anything.

```
#include<stdio.h>

void greatNum(int a, int b);    // function declaration

int main()
```

```

{
    int i, j;
    printf("Enter 2 numbers that you want to compare...");
    scanf("%d%d", &i, &j);
    greatNum(i, j);    // function call
    return 0;
}

```

```

void greatNum(int x, int y)    // function definition
{
    if(x > y) {
        printf("The greater number is: %d", x);
    }
    else {
        printf("The greater number is: %d", y);
    }
}

```

4.Function with arguments and a return value

This is the best type, as this makes the function completely independent of inputs and outputs, and only the logic is defined inside the function body.

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

```
int greatNum(int a, int b);    // function declaration
```

```

int main()
{
    int i, j, result;
    printf("Enter 2 numbers that you want to compare...");
    scanf("%d%d", &i, &j);
    result = greatNum(i, j); // function call
    printf("The greater number is: %d", result);
    return 0;
}

```

```

int greatNum(int x, int y)    // function definition
{
    if(x > y) {
        return x;
    }
    else {
        return y;
    }
}

```

Nesting of Functions

C language also allows nesting of functions i.e to use/call one function inside another function's body. We must be careful while using nested functions, because it may lead to infinite nesting.

```
function1()
{
    // function1 body here

    function2();

    // function1 body here
}
```

If function2() also has a call for function1() inside it, then in that case, it will lead to an infinite nesting. They will keep calling each other and the program will never terminate.

Not able to understand? Lets consider that inside the main() function, function1() is called and its execution starts, then inside function1(), we have a call for function2(), so the control of program will go to the function2(). But as function2() also has a call to function1() in its body, it will call function1(), which will again call function2(), and this will go on for infinite times, until you forcefully exit from program execution.

Recursion

Recursion is a special way of nesting functions, where a function calls itself inside it. We must have certain conditions in the function to break out of the recursion, otherwise recursion will occur infinite times.

```
function1()
{
    // function1 body
    function1();
    // function1 body
}
```

Example: Factorial of a number using Recursion

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

```
int factorial(int x);    //declaring the function
```

```
void main()
{
    int a, b;

    printf("Enter a number...");
```

```

scanf("%d", &a);
b = factorial(a);    //calling the function named factorial
printf("%d", b);
}

int factorial(int x) //defining the function
{
    int r = 1;
    if(x == 1)
        return 1;
    else
        r = x*factorial(x-1);    //recursion, since the function calls itself

    return r;
}

```

Similarly, there are many more applications of recursion in C language. Go to the programs section, to find out more programs using recursion.

Types of Function calls:

Functions are called by their names, we all know that, then what is this tutorial for? Well if the function does not have any arguments, then to call a function you can directly use its name. But for functions with arguments, we can call a function in two different ways, based on how we specify the arguments, and these two ways are:

1. Call by Value
2. Call by Reference

1.Call by Value

Calling a function by value means, we pass the values of the arguments which are stored or copied into the formal parameters of the function. Hence, the original values are unchanged only the parameters inside the function changes.

```

#include<stdio.h>

void calc(int x);

int main()
{
    int x = 10;
    calc(x);
    // this will print the value of 'x'
    printf("\nvalue of x in main is %d", x);
    return 0;
}

```



```

void calc(int x)
{
    // changing the value of 'x'
    x = x + 10 ;
    printf("value of x in calc function is %d ", x);
}

```

value of x in calc function is 20 value of x in main is 10

2.Call by Reference

In call by reference we pass the address(reference) of a variable as argument to any function. When we pass the address of any variable as argument, then the function will have access to our variable, as it now knows where it is stored and hence can easily update its value.

In this case the formal parameter can be taken as a **reference** or a **pointer**(don't worry about pointers, we will soon learn about them), in both the cases they will change the values of the original variable.

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

```
void calc(int *p);    // functin taking pointer as argument
```

```

int main()
{
    int x = 10;
    calc(&x);    // passing address of 'x' as argument
    printf("value of x is %d", x);
    return(0);
}

```

```

void calc(int *p)    //receiving the address in a reference pointer variable
{
    /*
        changing the value directly that is
        stored at the address passed
    */
    *p = *p + 10;
}

```

value of x is 20

UNIT – IV

STRUCTURES

UNIT IV

C Structures

Structure is a user-defined datatype in C language which allows us to combine data of different types together. Structure helps to construct a complex data type which is more meaningful. It is somewhat similar to an Array, but an array holds data of similar type only. But structure on the other hand, can store data of any type, which is practical more useful.

Defining a structure

struct keyword is used to define a structure. struct defines a new data type which is a collection of primary and derived datatypes.

Syntax:

```
struct [structure_tag]
{
    //member variable 1
    //member variable 2
    //member variable 3
    ...
}[structure_variables];
```

we start with the struct keyword, then it's optional to provide your structure a name, we suggest you to give it a name, then inside the curly braces, we have to mention all the member variables, which are nothing but normal C language variables of different types like int, float, array etc.

After the closing curly brace, we can specify one or more structure variables, again this is optional.

Note: The closing curly brace in the structure type declaration must be followed by a semicolon(;).

Example of Structure

```
struct Student
{
    char name[25];
    int age;
    char branch[10];
    // F for female and M for male
    char gender;
};
```

Here struct Student declares a structure to hold the details of a student which consists of 4 data fields, namely name, age, branch and gender. These fields are called **structure elements or members**.

Each member can have different datatype, like in this case, name is an array of char type and age is of int type etc. **Student** is the name of the structure and is called as the **structure tag**.

Structure variable declaration is similar to the declaration of any normal variable of any other datatype. Structure variables can be declared in following two ways: 1) Declaring Structure variables separately

```
struct Student
{
    char name[25];
    int age;
    char branch[10];
    //F for female and M for male
    char gender;
};
```

```
struct Student S1, S2;    //declaring variables of struct Student
```

2) Declaring Structure variables with structure definition

```
struct Student
{
    char name[25];
    int age;
    char branch[10];
    //F for female and M for male
    char gender;
}S1, S2;
```

Structure Initialization

Like a variable of any other datatype, structure variable can also be initialized at compile time.

```
struct Patient
{
    float height;
    int weight;
    int age;
};
```

```
struct Patient p1 = { 180.75 , 73, 23 };
```

Nested Structures

Nesting of structures, is also permitted in C language. Nested structures means, that one structure has another structure as member variable.

Example:

```
struct Student
{
    char[30] name;
```

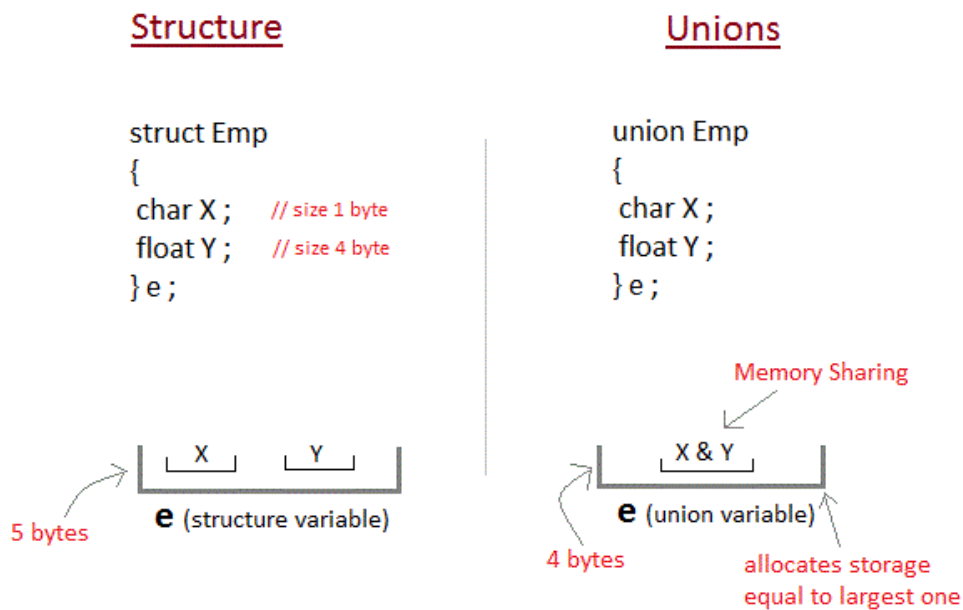
```

int age;
/* here Address is a structure */
struct Address
{
    char[50] locality;
    char[50] city;
    int pincode;
}addr;
};

```

Unions in C

Unions are conceptually similar to **structures**. The syntax to declare/define a union is also similar to that of a structure. The only difference is in terms of storage. In **structure** each member has its own storage location, whereas all members of **union** use a single shared memory location which is equal to the size of its largest data member.



This implies that although a **union** may contain many members of different types, **it cannot handle all the members at the same time**. A **union** is declared using the union keyword.

```

union item
{
    int m;
    float x;
    char c;
}It1;

```

This union contains three members each with a different data type. However only one of them can be used at a time. This is due to the fact that only one location is allocated for all the union variables, irrespective of their size.

Pointers

A Pointer in C language is a variable which holds the address of another variable of same data type.

Pointers are used to access memory and manipulate the address.

Pointers are one of the most distinct and exciting features of C language. It provides power and flexibility to the language. Although pointers may appear a little confusing and complicated in the beginning, but trust me, once you understand the concept, you will be able to do so much more with C language.

Address in C

Whenever a variable is defined in C language, a memory location is assigned for it, in which its value will be stored. We can easily check this memory address, using the & symbol.

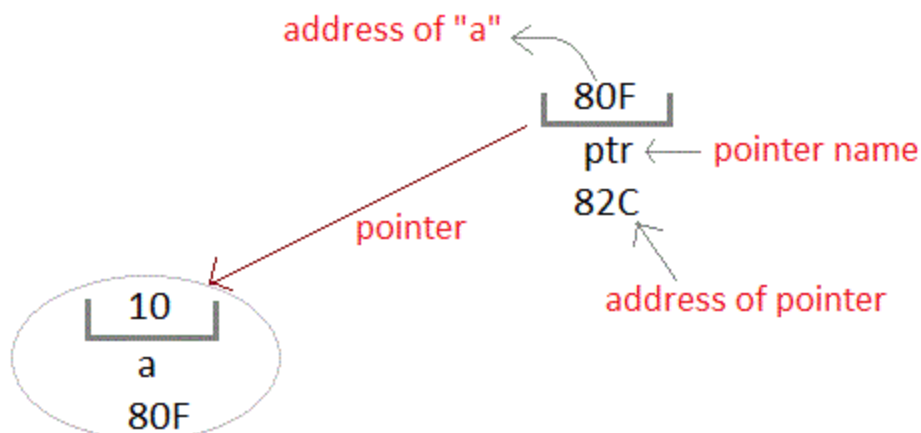
If var is the name of the variable, then &var will give its address.

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

```
void main()
{
    int var = 7;
    printf("Value of the variable var is: %d\n", var);
    printf("Memory address of the variable var is: %x\n", &var);
}
```

The variables which are used to hold memory addresses are called **Pointer variables**.

A **pointer** variable is therefore nothing but a variable which holds an address of some other variable. And the value of a **pointer variable** gets stored in another memory location.



Benefits of using pointers

Below we have listed a few benefits of using pointers:

1. Pointers are more efficient in handling Arrays and Structures.
2. Pointers allow references to function and thereby helps in passing of function as arguments to other functions.
3. It reduces length of the program and its execution time as well.
4. It allows C language to support Dynamic Memory management.

Declaring, Initializing and using a pointer variable in C

1. While declaring/initializing the pointer variable, * indicates that the variable is a pointer.
2. The address of any variable is given by preceding the variable name with Ampersand &.
3. The pointer variable stores the address of a variable. The declaration `int *a` doesn't mean that a is going to contain an integer value. It means that a is going to contain the address of a variable storing integer value.
4. To access the value of a certain address stored by a pointer variable, * is used. Here, the * can be read as '**value at**'.

SYNTAX

```
datatype *pointer_name;
```

EXAMPLE

```
int *ip // pointer to integer variable
float *fp; // pointer to float variable
double *dp; // pointer to double variable
char *cp; // pointer to char variable
```

Initialization of C Pointer variable

Pointer Initialization is the process of assigning address of a variable to a **pointer** variable. Pointer variable can only contain address of a variable of the same data type. In C language **address operator** & is used to determine the address of a variable. The & (immediately preceding a variable name) returns the address of the variable associated with it.

```
#include<stdio.h>

void main()
{
    int a = 10;
    int *ptr;    //pointer declaration
    ptr = &a;    //pointer initialization
}
```

Pointer to a Pointer in C(Double Pointer)

Pointers are used to store the address of other variables of similar datatype. But if you want to store the address of a pointer variable, then you again need a pointer to store it. Thus, when one pointer variable stores the address of another pointer variable, it is known as **Pointer to Pointer** variable or **Double Pointer**.

Syntax:

```
int **p1;
```

Here, we have used two indirection operator(*) which stores and points to the address of a pointer variable i.e, int *. If we want to store the address of this (double pointer) variable p1, then the syntax would become:

```
int ***p2
```

Simple program to represent Pointer to a Pointer

```
#include <stdio.h>

int main() {

    int a = 10;
    int *p1;    //this can store the address of variable a
    int **p2;
    /*
        this can store the address of pointer variable p1 only.
        It cannot store the address of variable 'a'
    */
}
```

```

p1 = &a;
p2 = &p1;

printf("Address of a = %u\n", &a);
printf("Address of p1 = %u\n", &p1);
printf("Address of p2 = %u\n\n", &p2);

// below print statement will give the address of 'a'
printf("Value at the address stored by p2 = %u\n", *p2);

printf("Value at the address stored by p1 = %d\n\n", *p1);

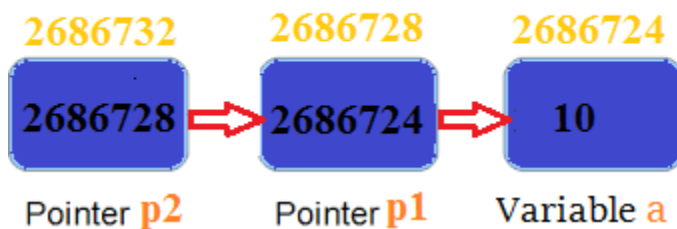
printf("Value of **p2 = %d\n", **p2); //read this *(*p2)

/*
   This is not allowed, it will give a compile time error-
   p2 = &a;
   printf("%u", p2);
*/
return 0;
}

```

Address of a = 2686724 Address of p1 = 2686728 Address of p2 = 2686732 Value at the address stored by p2 = 2686724 Value at the address stored by p1 = 10 Value of **p2 = 10

Explanation of the above program



- p1 pointer variable can only hold the address of the variable a (i.e Number of indirection operator(*)-1 variable). Similarly, p2 variable can only hold the address of variable p1. It cannot hold the address of variable a.
- *p2 gives us the value at an address stored by the p2 pointer. p2 stores the address of p1 pointer and value at the address of p1 is the address of variable a. Thus, *p2 prints address of a.

- `**p2` can be read as `*(**p2)`. Hence, it gives us the value stored at the address `*p2`. From above statement, you know `*p2` means the address of variable `a`. Hence, the value at the address `*p2` is 10. Thus, `**p2` prints 10.

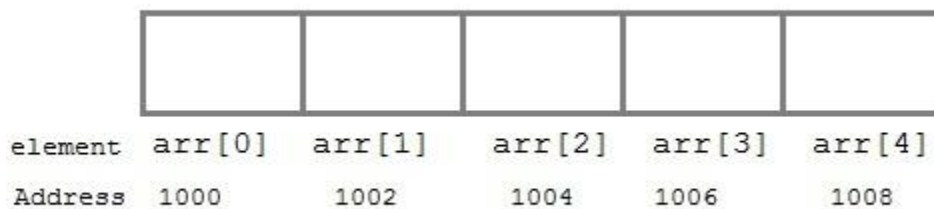
Pointer and Arrays in C

When an array is declared, compiler allocates sufficient amount of memory to contain all the elements of the array. Base address i.e address of the first element of the array is also allocated by the compiler.

Suppose we declare an array `arr`,

```
int arr[5] = { 1, 2, 3, 4, 5 };
```

Assuming that the base address of `arr` is 1000 and each integer requires two bytes, the five elements will be stored as follows:



Here variable `arr` will give the base address, which is a constant pointer pointing to the first element of the array, `arr[0]`. Hence `arr` contains the address of `arr[0]` i.e 1000. In short, `arr` has two purpose - it is the name of the array and it acts as a pointer pointing towards the first element in the array.

`arr` is equal to `&arr[0]` by default

We can also declare a pointer of type `int` to point to the array `arr`.

```
int *p;
p = arr;
// or,
p = &arr[0]; //both the statements are equivalent.
```

Now we can access every element of the array `arr` using `p++` to move from one element to another.

NOTE: You cannot decrement a pointer once incremented. `p--` won't work.

1.Pointer to Array

As studied above, we can use a pointer to point to an array, and then we can use that pointer to access the array elements. Lets have an example,

```
#include <stdio.h>

int main()
{
    int i;
    int a[5] = { 1, 2, 3, 4, 5};
    int *p = a;    // same as int*p = &a[0]
    for (i = 0; i < 5; i++)
    {
        printf("%d", *p);
        p++;
    }

    return 0;
}
```

In the above program, the pointer *p will print all the values stored in the array one by one. We can also use the Base address (a in above case) to act as a pointer and print all the values.

he generalized form for using pointer with an array,

`*(a+i)`

is same as:

`a[i]`

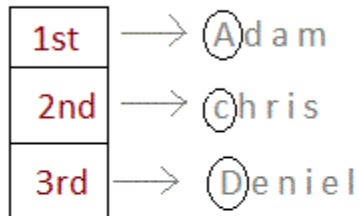
2.Array of Pointers

We can also have array of pointers. Pointers are very helpful in handling character array with rows of varying length.

```
char *name[3] = {
    "Adam",
    "chris",
    "Deniel"
};
//Now lets see same array without using pointer
char name[3][20] = {
    "Adam",
    "chris",
```

```
"Deniel"  
};
```

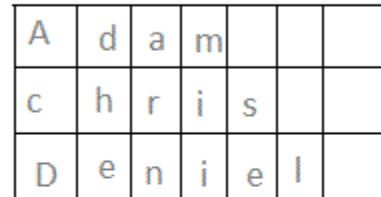
Using Pointer



```
char* name[3]
```

Only 3 locations for pointers, which will point to the first character of their respective strings.

Without Pointer



```
char name[3][20]
```

extends till 20 memory locations

In the second approach memory wastage is more, hence it is preferred to use pointer in such cases.

When we say memory wastage, it doesn't mean that the strings will start occupying less space, no, characters will take the same space, but when we define array of characters, a contiguous memory space is located equal to the maximum size of the array, which is a wastage, which can be avoided if we use pointers instead.

Pointer to Array of Structures in C

Like we have array of integers, array of pointers etc, we can also have array of structure variables. And to use the array of structure variables efficiently, we use **pointers of structure type**. We can also have pointer to a single structure variable, but it is mostly used when we are dealing with array of structure variables.

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

```
struct Book  
{  
    char name[10];  
    int price;  
}
```

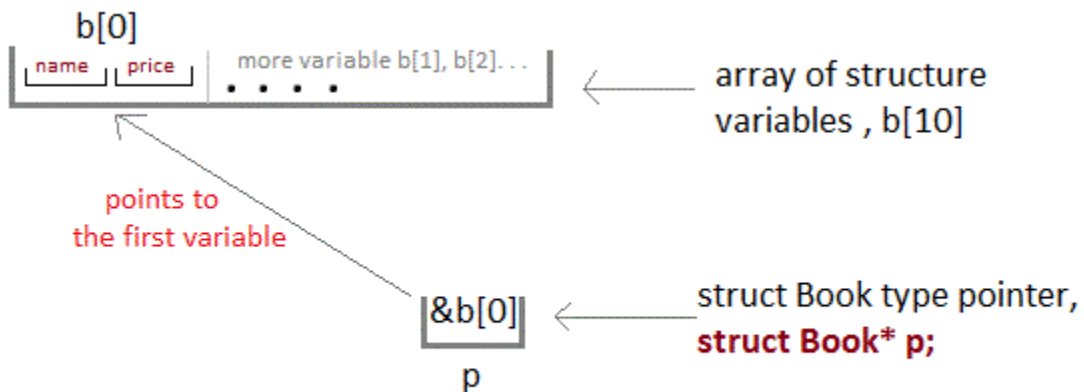
```

int main()
{
    struct Book a;    //Single structure variable
    struct Book* ptr; //Pointer of Structure type
    ptr = &a;

    struct Book b[10]; //Array of structure variables
    struct Book* p;    //Pointer of Structure type
    p = &b;

    return 0;
}

```



Accessing Structure Members with Pointer

To access members of structure using the structure variable, we used the dot . operator. But when we have a pointer of structure type, we use arrow -> to access structure members.

```

#include <stdio.h>

struct my_structure {
    char name[20];
    int number;
    int rank;
};

int main()
{
    struct my_structure variable = {"StudyTonight", 35, 1};

    struct my_structure *ptr;

```

```

ptr = &variable;

printf("NAME: %s\n", ptr->name);
printf("NUMBER: %d\n", ptr->number);
printf("RANK: %d", ptr->rank);

return 0;
}

```

NAME: StudyTonight NUMBER: 35 RANK: 1

Pointers as Function Argument in C

Pointer as a function parameter is used to hold addresses of arguments passed during function call. This is also known as **call by reference**. When a function is called by reference any change made to the reference variable will effect the original variable.

Example Time: Swapping two numbers using Pointer

```

#include <stdio.h>

void swap(int *a, int *b);

int main()
{
    int m = 10, n = 20;
    printf("m = %d\n", m);
    printf("n = %d\n\n", n);

    swap(&m, &n); //passing address of m and n to the swap function
    printf("After Swapping:\n\n");
    printf("m = %d\n", m);
    printf("n = %d", n);
    return 0;
}

/*
    pointer 'a' and 'b' holds and
    points to the address of 'm' and 'n'
*/
void swap(int *a, int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}

```

m = 10 n = 20 After Swapping: m = 20 n = 10

1.Functions returning Pointer variables

A function can also return a pointer to the calling function. In this case you must be careful, because local variables of function doesn't live outside the function. They have scope only inside the function. Hence if you return a pointer connected to a local variable, that pointer will be pointing to nothing when the function ends.

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

```
int* larger(int*, int*);
```

```
void main()
```

```
{
```

```
    int a = 15;
```

```
    int b = 92;
```

```
    int *p;
```

```
    p = larger(&a, &b);
```

```
    printf("%d is larger",*p);
```

```
}
```

```
int* larger(int *x, int *y)
```

```
{
```

```
    if(*x > *y)
```

```
        return x;
```

```
    else
```

```
        return y;
```

```
}
```

92 is larger

2.Pointer to functions

It is possible to declare a pointer pointing to a function which can then be used as an argument in another function. A pointer to a function is declared as follows,

```
type (*pointer-name)(parameter);
```

Here is an example :

```
int (*sum)(); //legal declaration of pointer to function
```

```
int *sum(); //This is not a declaration of pointer to function
```

A function pointer can point to a specific function when it is assigned the name of that function.

```
int sum(int, int);
int (*s)(int, int);
s = sum;
```

Here s is a pointer to a function sum. Now sum can be called using function pointer s along with providing the required argument values.

```
s (10, 20);
```

Example of Pointer to Function

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

```
int sum(int x, int y)
{
    return x+y;
}
```

```
int main( )
{
    int (*fp)(int, int);
    fp = sum;
    int s = fp(10, 15);
    printf("Sum is %d", s);

    return 0;
}
```

```
25
```

File Input/Output

A **file** represents a sequence of bytes on the disk where a group of related data is stored. File is created for permanent storage of data. It is a ready made structure.

In C language, we use a structure **pointer of file type** to declare a file.

```
FILE *fp;
```

C provides a number of functions that helps to perform basic file operations. Following are the functions,

Function	description
fopen()	create a new file or open a existing file
fclose()	closes a file
getc()	reads a character from a file
putc()	writes a character to a file
fscanf()	reads a set of data from a file
fprintf()	writes a set of data to a file
getw()	reads a integer from a file
putw()	writes a integer to a file
fseek()	set the position to desire point
ftell()	gives current position in the file
rewind()	set the position to the begining point

Opening a File or Creating a File

The fopen() function is used to create a new file or to open an existing file.

General Syntax:

```
*fp = FILE *fopen(const char *filename, const char *mode);
```

Here, *fp is the FILE pointer (FILE *fp), which will hold the reference to the opened(or created) file.

filename is the name of the file to be opened and **mode** specifies the purpose of opening the file. Mode can be of following types,

mode	description
r	opens a text file in reading mode
w	opens or create a text file in writing mode.
a	opens a text file in append mode
r+	opens a text file in both reading and writing mode
w+	opens a text file in both reading and writing mode
a+	opens a text file in both reading and writing mode
rb	opens a binary file in reading mode
wb	opens or create a binary file in writing mode
ab	opens a binary file in append mode
rb+	opens a binary file in both reading and writing mode
wb+	opens a binary file in both reading and writing mode
ab+	opens a binary file in both reading and writing mode

Closing a File

The fclose() function is used to close an already opened file.

General Syntax :

```
int fclose( FILE *fp);
```

Here fclose() function closes the file and returns **zero** on success, or **EOF** if there is an error in closing the file. This **EOF** is a constant defined in the header file **stdio.h**.

Input/Output operation on File

In the above table we have discussed about various file I/O functions to perform reading and writing on file. getc() and putc() are the simplest functions which can be used to read and write individual characters to a file.

```
#include<stdio.h>

int main()
{
    FILE *fp;
    char ch;
    fp = fopen("one.txt", "w");
    printf("Enter data...");
    while( (ch = getchar()) != EOF) {
        putc(ch, fp);
    }
    fclose(fp);
    fp = fopen("one.txt", "r");

    while( (ch = getc(fp)) != EOF)
        printf("%c",ch);

    // closing the file pointer
    fclose(fp);

    return 0;
}
```

1. Reading and Writing to File using fprintf() and fscanf()

```
#include<stdio.h>

struct emp
{
    char name[10];
    int age;
};
```

```

void main()
{
    struct emp e;
    FILE *p,*q;
    p = fopen("one.txt", "a");
    q = fopen("one.txt", "r");
    printf("Enter Name and Age:");
    scanf("%s %d", e.name, &e.age);
    fprintf(p, "%s %d", e.name, e.age);
    fclose(p);
    do
    {
        fscanf(q, "%s %d", e.name, e.age);
        printf("%s %d", e.name, e.age);
    }
    while(!feof(q));
}

```

In this program, we have created two FILE pointers and both are referring to the same file but in different modes.

fprintf() function directly writes into the file, while fscanf() reads from the file, which can then be printed on the console using standard printf() function.

2. Difference between Append and Write Mode

Write (w) mode and Append (a) mode, while opening a file are almost the same. Both are used to write in a file. In both the modes, new file is created if it doesn't exist already.

The only difference they have is, when you **open** a file in the **write** mode, the file is reset, resulting in deletion of any data already present in the file. While in **append** mode this will not happen. Append mode is used to append or add data to the existing data of file (if any). Hence, when you open a file in Append (a) mode, the cursor is positioned at the end of the present data in the file.

3. Reading and Writing in a Binary File

A Binary file is similar to a text file, but it contains only large numerical data. The Opening modes are mentioned in the table for opening modes above.

fread() and fwrite() functions are used to read and write in a binary file.

fwrite(data-element-to-be-written, size_of_elements, number_of_elements, pointer-to-file);

fread() is also used in the same way, with the same arguments like fwrite() function. Below mentioned is a simple example of writing into a binary file

```
const char *mytext = "The quick brown fox jumps over the lazy dog";
```

```
FILE *bfp= fopen("test.txt", "wb");
if (bfp)
{
    fwrite(mytext, sizeof(char), strlen(mytext), bfp);
    fclose(bfp);
}
```

fseek(), ftell() and rewind() functions

- **fseek():** It is used to move the reading control to different positions using fseek function.
- **ftell():** It tells the byte location of current position of cursor in file pointer.
- **rewind():** It moves the control to beginning of the file.

Some File Handling Program Examples

- List all the Files present in a Directory
- Read Content of a File and Display it on screen
- Finding Size of a File
- Create a File and store Information in it
- Reverse the Content of File and Print it
- Copy Content of one File into Another File

UNIT – V

Dynamic Memory Allocation in C

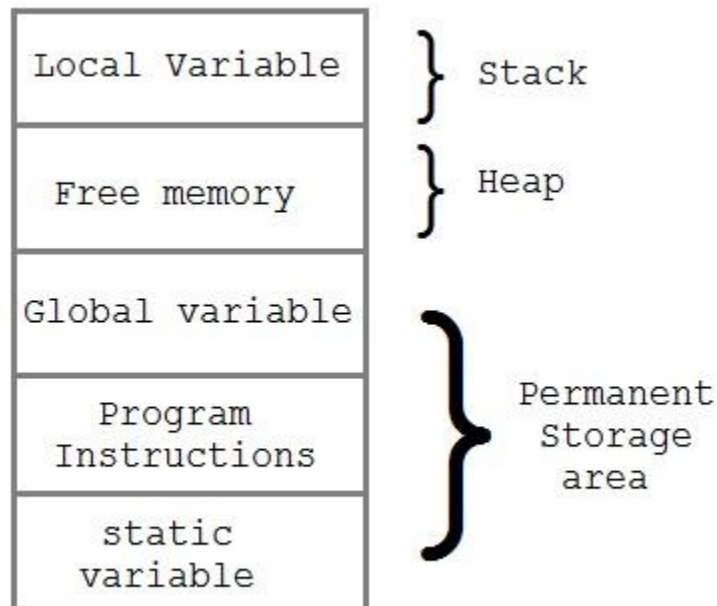
The process of allocating memory at runtime is known as **dynamic memory allocation**. Library routines known as **memory management functions** are used for allocating and freeing memory during execution of a program. These functions are defined in **stdlib.h** header file.

Function	Description
malloc()	allocates requested size of bytes and returns a void pointer pointing to the first byte of the allocated space
calloc()	allocates space for an array of elements, initialize them to zero and then returns a void pointer to the memory
free	releases previously allocated memory
realloc	modify the size of previously allocated space

Memory Allocation Process

Global variables, static variables and program instructions get their memory in **permanent** storage area whereas **local** variables are stored in a memory area called **Stack**.

The memory space between these two region is known as **Heap** area. This region is used for dynamic memory allocation during execution of the program. The size of heap keep changing.



Allocating block of Memory

malloc() function is used for allocating block of memory at runtime. This function reserves a block of memory of the given size and returns a **pointer** of type void. This means that we can assign it to any type of pointer using typecasting. If it fails to allocate enough space as specified, it returns a NULL pointer.

Syntax:

```
void* malloc(byte-size)
```

Time for an Example: malloc()

```
int *x;  
x = (int*)malloc(50 * sizeof(int)); //memory space allocated to variable x  
free(x); //releases the memory allocated to variable x
```

calloc() is another memory allocation function that is used for allocating memory at runtime. calloc function is normally used for allocating memory to derived data types such as **arrays** and **structures**. If it fails to allocate enough space as specified, it returns a NULL pointer.

Syntax:

```
void *calloc(number of items, element-size)
```

Time for an Example: calloc()

```
struct employee  
{  
    char *name;  
    int salary;  
};  
typedef struct employee emp;  
emp *e1;  
e1 = (emp*)calloc(30,sizeof(emp));
```

realloc() changes memory size that is already allocated dynamically to a variable.

Syntax:

```
void* realloc(pointer, new-size)
```

Time for an Example: realloc()

```
int *x;  
x = (int*)malloc(50 * sizeof(int));
```

```
x = (int*)realloc(x,100); //allocated a new memory to variable x
```

Difference between malloc() and calloc()

calloc()	malloc()
calloc() initializes the allocated memory with 0 value.	malloc() initializes the allocated memory with garbage values.
Number of arguments is 2	Number of argument is 1
Syntax : (cast_type *)calloc(blocks , size_of_block);	Syntax : (cast_type *)malloc(Size_in_bytes);

Program to represent Dynamic Memory Allocation(using calloc())

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

```
int main()
{
    int i, n;
    int *element;

    printf("Enter total number of elements: ");
    scanf("%d", &n);

    /*
     returns a void pointer(which is type-casted to int*)
     pointing to the first block of the allocated space
    */
    element = (int*) calloc(n,sizeof(int));

    /*
     If it fails to allocate enough space as specified,
     it returns a NULL pointer.
    */
    if(element == NULL)
    {
        printf("Error.Not enough space available");
        exit(0);
    }

    for(i = 0; i < n; i++)
    {
        /*
         storing elements from the user
        */
    }
}
```

```

        in the allocated space
        */
        scanf("%d", element+i);
    }
    for(i = 1; i < n; i++)
    {
        if(*element > *(element+i))
        {
            *element = *(element+i);
        }
    }

    printf("Smallest element is %d", *element);

    return 0;
}

```

Enter total number of elements: 5 4 2 1 5 3 Smallest element is 1

Allocating block of Memory

malloc() function is used for allocating block of memory at runtime. This function reserves a block of memory of the given size and returns a **pointer** of type void. This means that we can assign it to any type of pointer using typecasting. If it fails to allocate enough space as specified, it returns a NULL pointer.

Syntax:

```
void* malloc(byte-size)
```

Time for an Example: malloc()

```
int *x;
x = (int*)malloc(50 * sizeof(int)); //memory space allocated to variable x
free(x); //releases the memory allocated to variable x

```

calloc() is another memory allocation function that is used for allocating memory at runtime. calloc function is normally used for allocating memory to derived data types such as **arrays** and **structures**. If it fails to allocate enough space as specified, it returns a NULL pointer.

Syntax:

```
void *calloc(number of items, element-size)
```

Time for an Example: calloc()

```
struct employee
```

```

{
    char *name;
    int salary;
};
typedef struct employee emp;
emp *e1;
e1 = (emp*)calloc(30,sizeof(emp));

```

realloc() changes memory size that is already allocated dynamically to a variable.

Syntax:

```
void* realloc(pointer, new-size)
```

Time for an Example: realloc()

```

int *x;
x = (int*)malloc(50 * sizeof(int));
x = (int*)realloc(x,100); //allocated a new memory to variable x

```

Difference between malloc() and calloc()

calloc()	malloc()
calloc() initializes the allocated memory with 0 value.	malloc() initializes the allocated memory with garbage values.
Number of arguments is 2	Number of argument is 1
Syntax : (cast_type *)calloc(blocks , size_of_block);	Syntax : (cast_type *)malloc(Size_in_bytes);

Program to represent Dynamic Memory Allocation(using calloc())

```

#include <stdio.h>
#include <stdlib.h>

int main()
{
    int i, n;
    int *element;

    printf("Enter total number of elements: ");
    scanf("%d", &n);

    /*

```



```

    returns a void pointer(which is type-casted to int*)
    pointing to the first block of the allocated space
*/
element = (int*) calloc(n,sizeof(int));

/*
    If it fails to allocate enough space as specified,
    it returns a NULL pointer.
*/
if(element == NULL)
{
    printf("Error.Not enough space available");
    exit(0);
}

for(i = 0; i < n; i++)
{
    /*
        storing elements from the user
        in the allocated space
    */
    scanf("%d", element+i);
}
for(i = 1; i < n; i++)
{
    if(*element > *(element+i))
    {
        *element = *(element+i);
    }
}

printf("Smallest element is %d", *element);

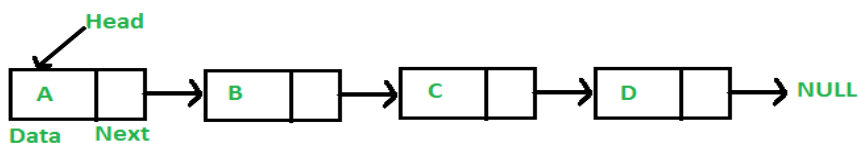
return 0;
}

```

Enter total number of elements: 5 4 2 1 5 3 Smallest element is 1

Linked List

Like arrays, Linked List is a linear data structure. Unlike arrays, linked list elements are not stored at a contiguous location; the elements are linked using pointers.



Definition of Linked List

Arrays can be used to store linear data of similar types, but arrays have the following limitations.

- 1) The size of the arrays is fixed: So we must know the upper limit on the number of elements in advance. Also, generally, the allocated memory is equal to the upper limit irrespective of the usage.
- 2) Inserting a new element in an array of elements is expensive because the room has to be created for the new elements and to create room existing elements have to be shifted.

Advantages over arrays

- 1) Dynamic size
- 2) Ease of insertion/deletion

Drawbacks:

- 1) Random access is not allowed. We have to access elements sequentially starting from the first node. So we cannot do binary search with linked lists efficiently with its default implementation. Read about it [here](#).
- 2) Extra memory space for a pointer is required with each element of the list.
- 3) Not cache friendly. Since array elements are contiguous locations, there is locality of reference which is not there in case of linked lists.

Representation:

A linked list is represented by a pointer to the first node of the linked list. The first node is called the head. If the linked list is empty, then the value of the head is NULL.

Each node in a list consists of at least two parts:

- 1) data
- 2) Pointer (Or Reference) to the next node

Preprocessor:

C Preprocessor is not a part of the compiler, but is a separate step in the compilation process. In simple terms, a C Preprocessor is just a text substitution tool and it instructs the compiler to do required pre-processing before the actual compilation. We'll refer to the C Preprocessor as CPP.

All preprocessor commands begin with a hash symbol (#). It must be the first nonblank character, and for readability, a preprocessor directive should begin in the first column. The following section lists down all the important preprocessor directives –

Sr.No.	Directive & Description
1	#define Substitutes a preprocessor macro.
2	#include Inserts a particular header from another file.
3	#undef Undefines a preprocessor macro.

4	#ifdef Returns true if this macro is defined.
5	#ifndef Returns true if this macro is not defined.
6	#if Tests if a compile time condition is true.
7	#else The alternative for #if.
8	#elif #else and #if in one statement.
9	#endif Ends preprocessor conditional.
10	#error Prints error message on stderr.
11	#pragma Issues special commands to the compiler, using a standardized method.

Preprocessors Examples

Analyze the following examples to understand various directives.

```
#define MAX_ARRAY_LENGTH 20
```

This directive tells the CPP to replace instances of `MAX_ARRAY_LENGTH` with `20`. Use `#define` for constants to increase readability.

```
#include <stdio.h>
#include "myheader.h"
```

These directives tell the CPP to get `stdio.h` from **System Libraries** and add the text to the current source file. The next line tells CPP to get `myheader.h` from the local directory and add the content to the current source file.

```
#undef FILE_SIZE
#define FILE_SIZE 42
```

It tells the CPP to undefine existing `FILE_SIZE` and define it as `42`.

```
#ifndef MESSAGE
#define MESSAGE "You wish!"
#endif
```

It tells the CPP to define `MESSAGE` only if `MESSAGE` isn't already defined.

```
#ifdef DEBUG
/* Your debugging statements here */
```