

FTBM 351
IT Applications in Food Industry
STUDY MATERIAL



Ms. K. Suseela
Assistant Professor
Department of Food Trade and Business Management
College of Food Science and Technology
Bapatla



FACULTY OF FOOD SCIENCE & TECHNOLOGY
Acharya N G Ranga Agricultural University
Rajendranagar, Hyderabad – 500 030

FTBM 351

2(1+1)

IT APPLICATIONS IN FOOD INDUSTRY**Objective**

The main objective of introducing this subject in the degree course of food technology is to expose the student with fundamental knowledge on software of computers as specially to 'C' Programming. It will also impart knowledge related to the applications of computation in food industries

LECTURE OUTLIENES

1. Course No : FTBM 351
2. Title : IT Applications in Food Industry
3. Credit Hours : 2(1+1)
4. General Objectives: :
 - Able to know about "The necessity of Software & their applications in Food Industries"
 - &
 - Able to Implement the Programs in 'C' to perform various operations that are related to Food Industries
5. Specific Objectives :
 - a) Theory:
 - ✚ Able to know about the various steps which are related to computer and Software and their application in Food Industries.
 - ✚ Able to know about the various steps which are necessary to implement the programs in 'C'.
 - b) Practical:
 - ✚ Able to know about "How to perform the various operations which are related to Food Industries in Microsoft Office Tools(Ex: MS-Excel, MS-Power Point...etc..,)
 - ✚ Able to know about "How to build the program to perform various mathematical, logical Operations in 'C' which are related to Food Industry.

A) Theory Lecture Outlines:

- Lecture :1 Computerization, Importance of Computerization in food industry and IT applications in food industries.
- Lecture :2 Computer operating environments and information system for various types of food industries.
Introduction to a Barcharts and Piecharts &
The procedure to develop a barchats and piecharts on given Data.
- Lecture :3 Introduction to Software & Programming Languages, Properties, Differences of an Algorithm and Flowcharts, Advantages and disadvantages of Flowcharts & Algorithms.
- Lecture :4 Introduction, Fundamentals & advantages of 'C'
Steps in learning 'C' (Character set, Identifiers, Keywords)
Steps in learning 'C' (Data types, Constants, Variables, Escape sequences)

- Lecture :6 Steps in learning 'C' (Operators, Statements)
Steps in learning 'C' (Header Files, Input & Output functions:
Formatted I/O functions Unformatted I/O functions)
- Lecture :7 Basic Structure of a simple 'C' program
- Lecture :8 Decision Making/Control Statements
- Lecture :9 Branching, Concept of Looping & Looping statements
- Lecture :10 Concept of Arrays & Types of Arrays(Single, Double and Multi dimensional Arrays)
- Lecture :11 Concept of Functions (Defining a function & Function Prototypes,
Types of functions: Library functions & User defined functions)
- Lecture :12 Concept of various types of User Defined Functions(i.e., About 4 types)
- Lecture :13 Concept of a String Library Functions
- Lecture :14 Concept of Pointers, Structures & Unions
- Lecture :15 Introduction to a Data Structures
Types of Data Structures (Primary & Secondary Data Structures)
Concept of a Linked Lists, Types of a Linked Lists
&
Basic operations on linked Lists.
- Lecture :16 Concept of Stacks & Operations on Stacks (PUSH & POP Operations)
Concept of Queues & types of Queues
Operations on a Queue (ENQUEUE & DEQUEUE Operations)

B) Practical Lecture Outlines:

- Practical :1 Application of MS Excel to solve the problems of food technology
Introduction to a C compiler & How to handle the C compiler
(Controllers used in C Compiler)
- Practical :2 Statistical quality controls of food Developing and executing simple C programs
(By using various operators used in 'C').
- Practical :3 Sensory Evaluation of food
Developing and executing simple 'C' programs
(By using some mathematical & logical operation)
- Practical :4 Chemical kinetics in Food processing
Developing and executing simple 'C' programs
(By using Control statements: if, if-else, multiple if-else)
- Practical :5 Use of Word Processing software (MS- Power Point) for creating reports and presentation
Developing and executing simple 'C' programs
(By using Control statements: nested if's, conditional operator, and switch statements)
- Practical :6 Familization with the application of computer in food industries
Developing and executing simple 'C' programs

(By using loops: while, do- while loops)

Practical :7 Milk plant, dairy units, fruit and vegetable processing unit familization with software related to food industries.

Developing and executing simple 'C' programs (By using loops: for loop)

Practical :8 Ergonamics application in the Food industries.

Developing and executing simple 'C' programs

(By using arrays: single (or) one dimensional arrays)

Practical :9 Developing and executing simple 'C' programs

(By using arrays: two dimensional arrays)

Practical :10 Developing and executing simple 'C' programs

(By using four types of Functions)

Practical :11 Developing and executing simple 'C' programs

(By using a string functions:strlen(), strrev(), strcpy())

Practical :12 Developing and executing simple 'C' programs

(By using a string functions: strcat(), strlwr(),strupr())

Practical :13 Developing and executing simple 'C' programs (By using pointers)

Practical :14 Developing and executing simple 'C' programs (By using structures, unions)

Practical :15 Developing and executing 'C' programs

(By using Linked lists & operations: Insertion & Deletion of a nodes to and from a linked lists respectively)

Developing and executing 'C' programs

(By using Stack & operations: PUSH & POP By using Queues & operations: ENQUEUE & DEQUEUE)

Practical :16 Final Practical examination

Reference books:

1. Let us 'C' – Yeswanth Kanethkar
2. Computer Programming in 'C' – E. Balaguruswamy
3. Data Structures – Mark Allen Waise
3. M. S Excel 2000 - Microsoft Corp.
4. M. S. Office – Microsoft Corp
5. Computer concepts for Agri Business concepts – M.V. Verton, AVI Pub. Corp, West Port, USA.

-----OoO-----

LECTURE 1:

COMPUTERIZATION, IMPORTANCE OF COMPUTERIZATION IN FOOD INDUSTRY
&
IT APLLICAITONS IN FOOD INDUSTRIES

- ❖ Introduction to various software for their application in food technology

Application of MS Excel to solve the problems of Food Technology:

a) Chemical kinetics in food processing:

- Determining rate constant of zero order reaction
- First order rate constant and half life of reactions
- Determining energy of activation of vitamin degradation during food storage
- Rates of Enzymes catalyzed reaction

b) Microbial distraction in thermal processing of food

- Determining decimal reduction time from microbial survival data
- Thermal resistance factor, Z-values in thermal processing of food
- Sampling to ensure that a lot is not contaminated with more than a given percentage

c) Statistical quality control

- Probability of occurrence in normal distribution
- Using binomial distribution to determine probability of occurrence
- Probability of defective items in a sample obtained from large lot

d) Sensory evaluation of food

- Statistical descriptors of a population estimated from sensory data obtained from a sample
- Analysis of variance

- * One factor, completely randomized design
- * For two factor design without replication

- Use of linear regression in analyzing sensory data




e) Mechanical transport of liquid food

- Measuring viscosity of liquid food using a capillary tube viscometer

f) Solving simultaneous equations in designing multiple effect evaporators while using matrix algebra available in excel

LECTURE 2:

INTRODUCTION TO A BARCHARTS AND PIECHARTS
&
THE PROCEDURE TO DEVELOP A BARCHARTS AND PIECHARTS ON GIVEN DATA

- A spreadsheet is essentially a matrix of rows and columns. Consider a sheet of paper on which horizontal and vertical lines are drawn to yield a rectangular grid. The grid namely a cell, is the result of the intersection of a row with a column. Such a structure is called a Spreadsheet.
 - A spreadsheet package contains electronic equivalent of a pen, an eraser and large sheet of paper with vertical and horizontal lines to give rows and columns. The cursor position uniquely shown in dark mode indicates where the pen is currently pointing. We can enter text (or) numbers at any position on the worksheet. We can enter a formula in a cell where we want to perform a calculation and results are to be displayed. A powerful recalculation facility jumps into action each time we update the cell contents with new data.
 - MS-Excel is the most powerful spreadsheet package brought by Microsoft.
- The three main components of this package are
-  Electronic spreadsheet
 -  Database management
 -  Generation of Charts
- MS-Excel is the most powerful spreadsheet package brought by Microsoft.
 - Each workbook provides 3 worksheets with facility to increase the number of sheets. Each sheet provides 256 columns and 65536 rows to work with.
 - Though the spreadsheet packages were originally designed for accountants, they have become popular with almost everyone working with figures.
 - Sales executives, book-keepers, officers, students, research scholars, investors bankers etc, almost any one find some form of application for it.

CHARTS:

- A chart is a graphical representation of the data in your worksheet. You can crate an embedded charts, which appears on the worksheet beside the data, (or) You can crate a chart sheet as a separate sheet in the workbook so that is can be displayed apart from its associated data.
 - Whichever method you choose, your chart data is automatically linked to the worksheet from which it was created. If you change the data on the worksheet, the chart will change accordingly.
- Excel offers many different charts types, each of which has several subtypes (or) variations.

Column	Bar	Line	Pie	XY (Scatter)	Area	Doughnut
Radar	Surface	Bubble	Stock	Cone	Cylinder	
Pyramid						

- An Excel workbook called **Data_for_Charts.xls** has been prepared for you to use while working through this Guide. It contains data that can readily be displayed in charts form.
- Sometimes the most difficult decision is choosing the appropriate chart type for the data. We have focused on a 2D Column chart because our table data lends itself well to this type of chart.

- Most data can be represented well in a column, bar, line, (or) area chart.
- Three dimensional charts tend to be a bit more difficult to read, but a 3D column or area chart is quite readable.
- A ribbon chart may be more difficult to read and suitable for impact rather than readability. A pie chart is a special case in which only one series is plotted.
- The XY chart is suitable for data when there is a dependency such as time versus distance (km/hr, for instance).
- Radar charts are used in medical applications, and a stock chart type may be suitable for some financial and engineering applications.

BAR CHARTS:

- Bar charts illustrate comparisons between items. Although they are similar to column charts, the categories on a bar charts are displayed vertically and the values are organized horizontally.
- This concentrates on comparing values and places less emphasis on time. A stacked bar chart shows the relationship of individual items to the whole.

PIE CHARTS:

- A Pie chart can show one data series. It compares the size of individual items with the sum of them all.

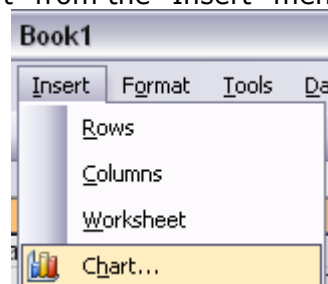
Chart Wizard:

Creating a BAR Chart:

- Excel allows you to create basic – to – intermediate charts based off of information and data within your spreadsheets. Let's create a column chart from the student grade data from before. First, highlight the data.

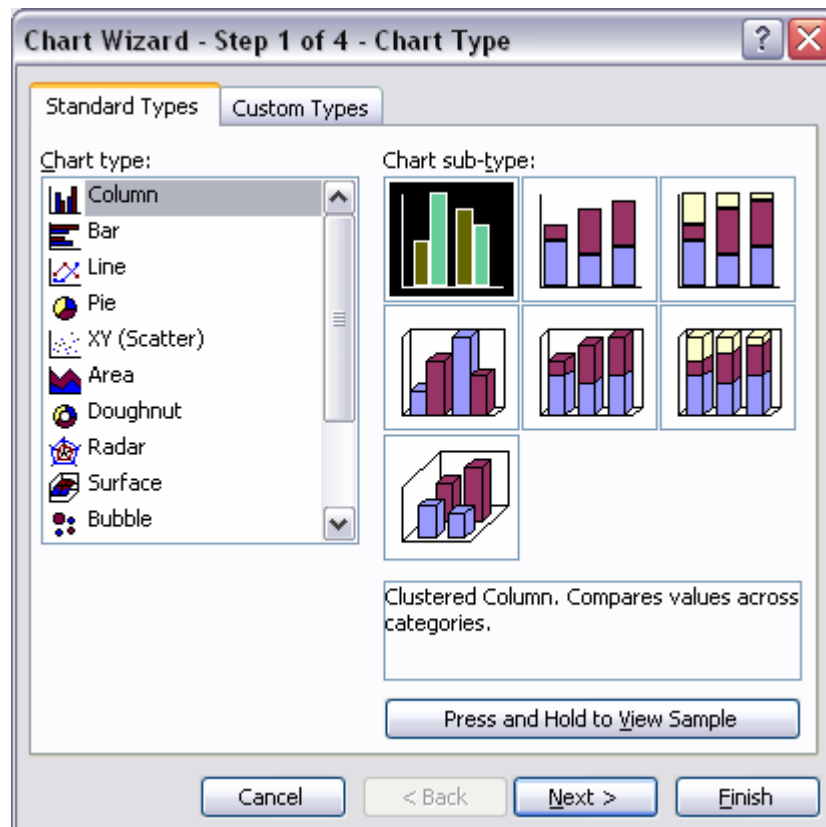
	A	B	C	D	E
1		Grade 1	Grade 2	Grade 3	
2	Student 3	90	100	100	96.66666667
3	Student 1	98	100	80	92.66666667
4	Student 4	86	88	90	88
5	Student 2	78	90	85	84.33333333
6	Student 5	0	0	0	0

Next, select "Chart" from the "Insert" menu.



A new window will appear asking which type of chart you would like to create.

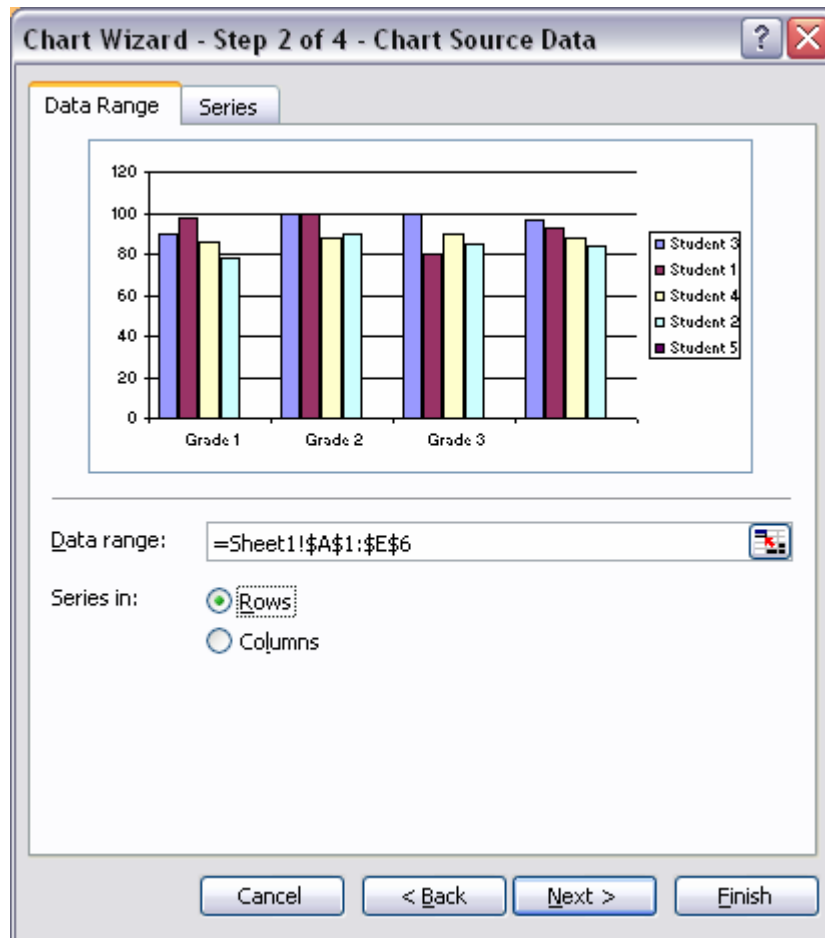
For this example, let's do a basic pie chart. Select "Column" from the "Chart Type" on the left side, and pick the first sub-type on the right (a normal, 2D column chart).



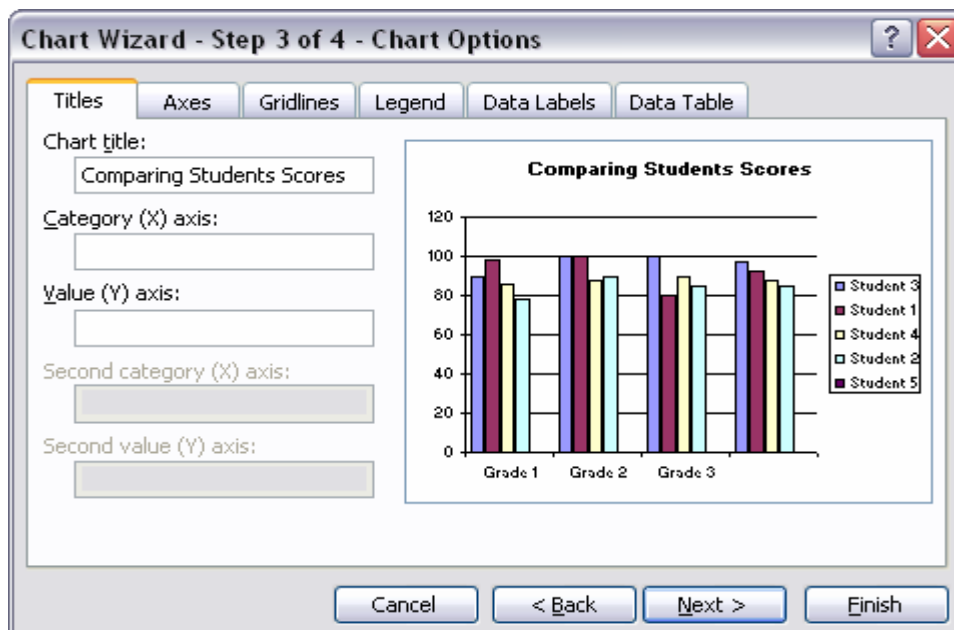
Click "Next." In this window, you'll be asked to select your "data range"; this is the area of your spreadsheet that you wish to generate a chart from. Since you've already selected the area before, it should already be entered into the appropriate area.

"Series in" allows you to choose by which value you want to arrange the chart.

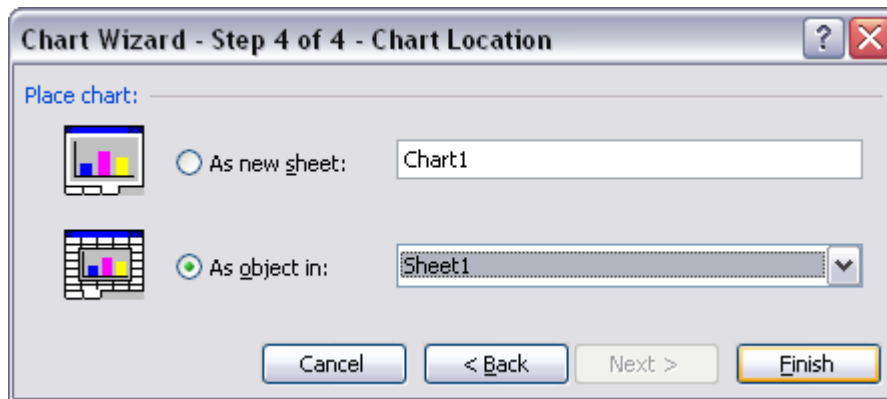
Let's arrange it by rows; this will break it down by "Grade" (such as Test 1, Test 2, etc.) and comparing the student scores next to each other.



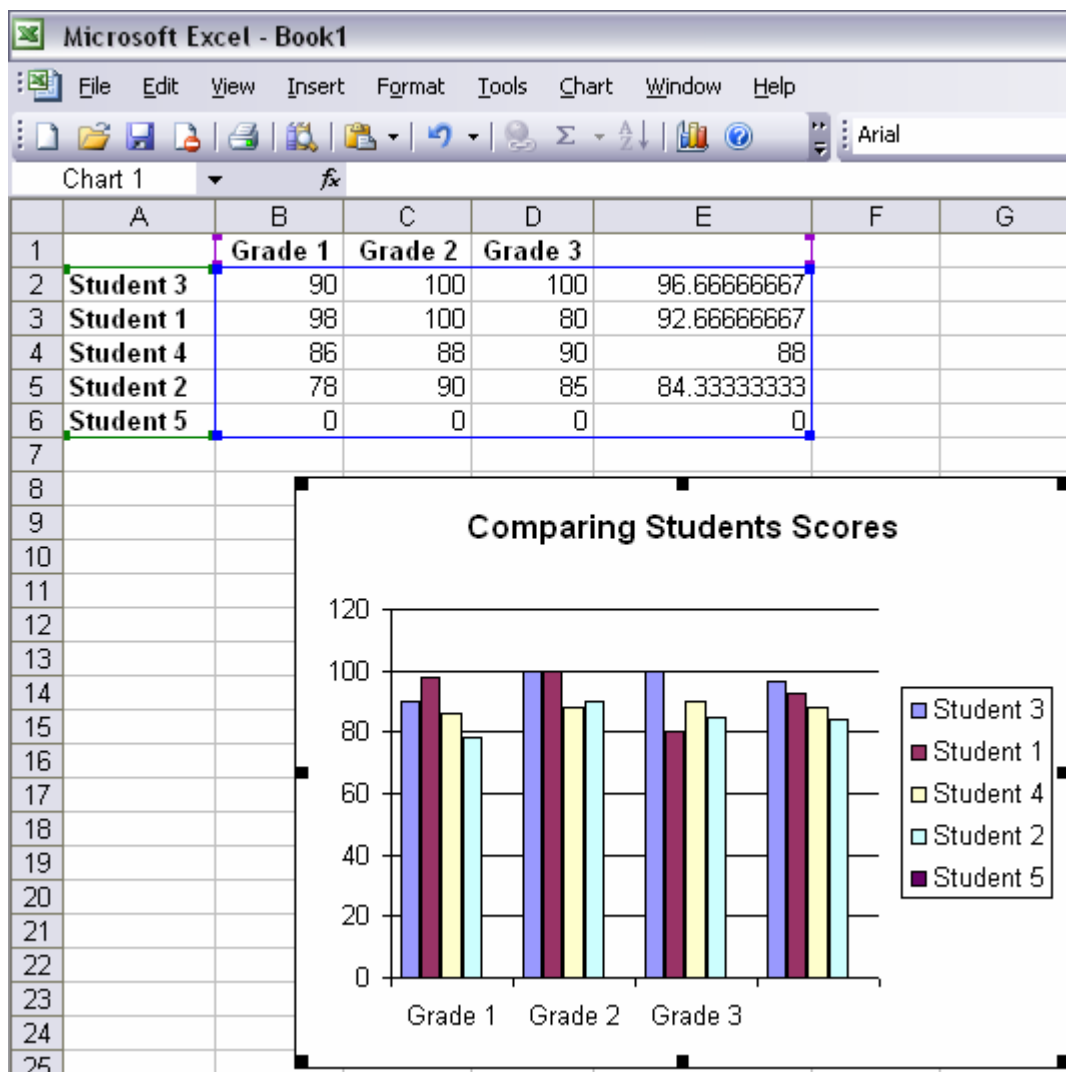
Click "Next." In step three you can give the chart a name ("Chart Title"), label the X and/or Y axis, etc.



Click "Next." The final step will ask whether you want the chart as an object in your current spreadsheet or in a new one; generally, you will place it within the same spreadsheet.

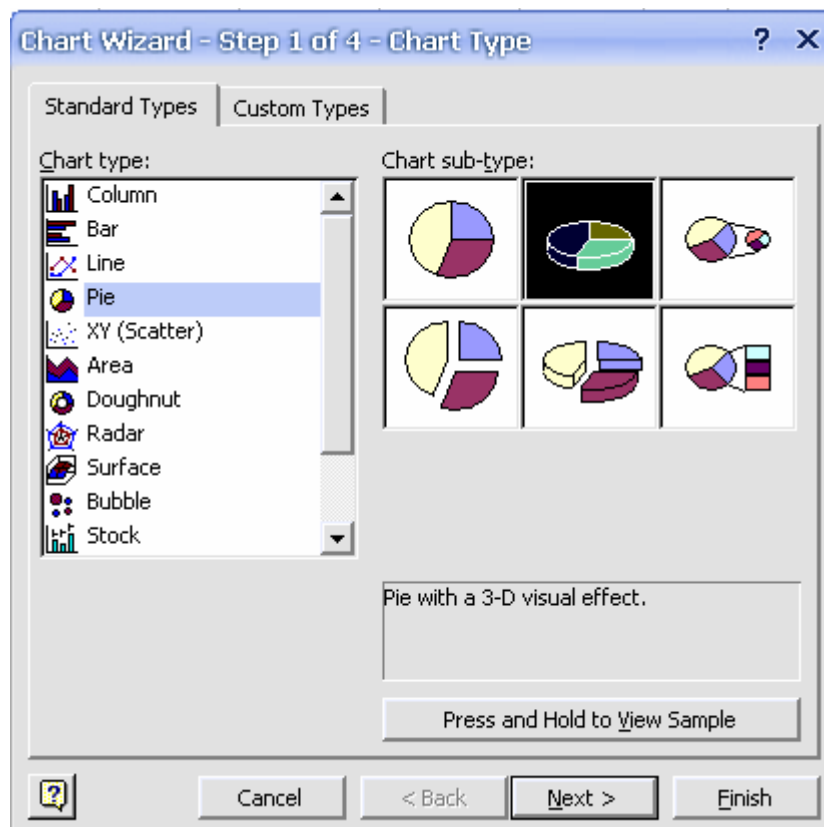


Click "Finish," and your chart will appear in your spreadsheet!

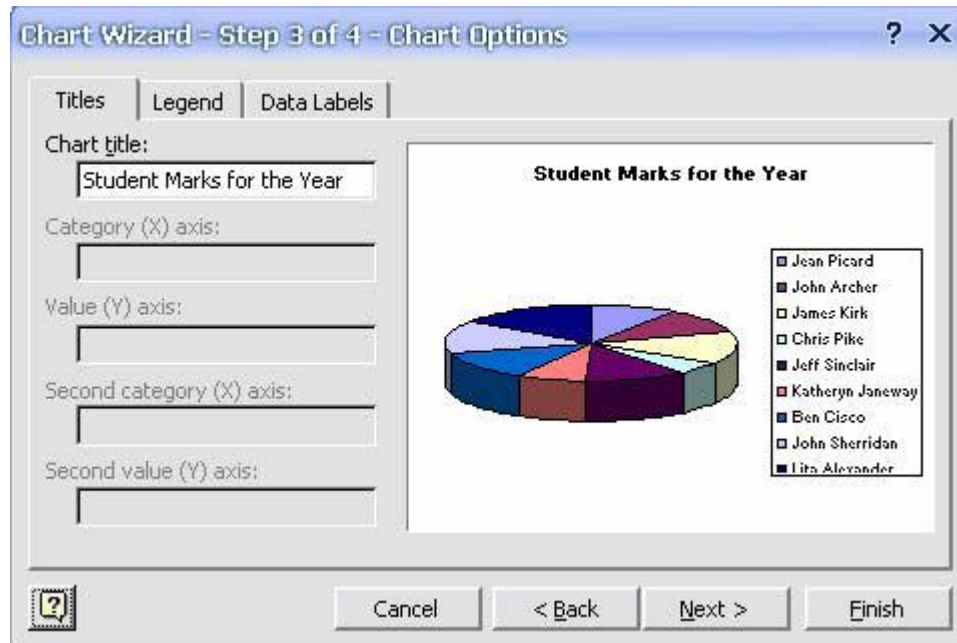


Creating a Pie Chart:

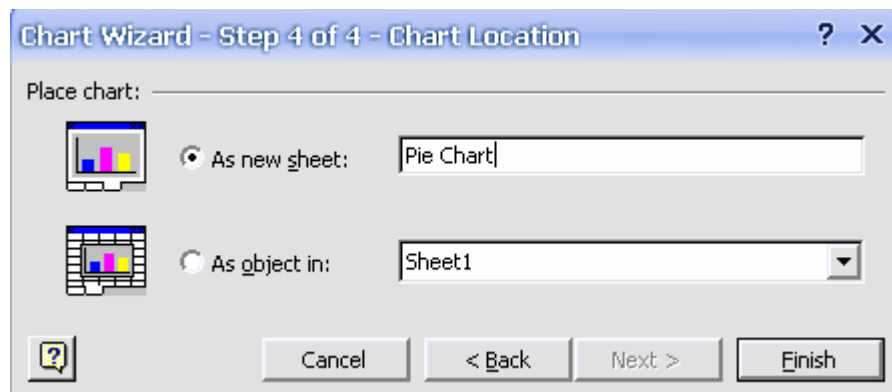
- One of the differences between a pie chart and a normal chart is that a pie chart will only show data for one series at a time.
 - In the following exercise, we'll create a pie chart from the year's data.
- We need to begin by selecting the names of the students (to be used as labels) and the data itself.
- 1) Select the cells with the student names including the student heading (A5:A14). Since we need to select more than one group of cells we'll need to use the [Ctrl] key.
 - 2) Hold down [Ctrl] and select the cells with the year data (F5:F14) A5:A14 and F5:F14 should now be selected.
 - 3) Click the chart wizard icon to begin creating the chart from the selected data.



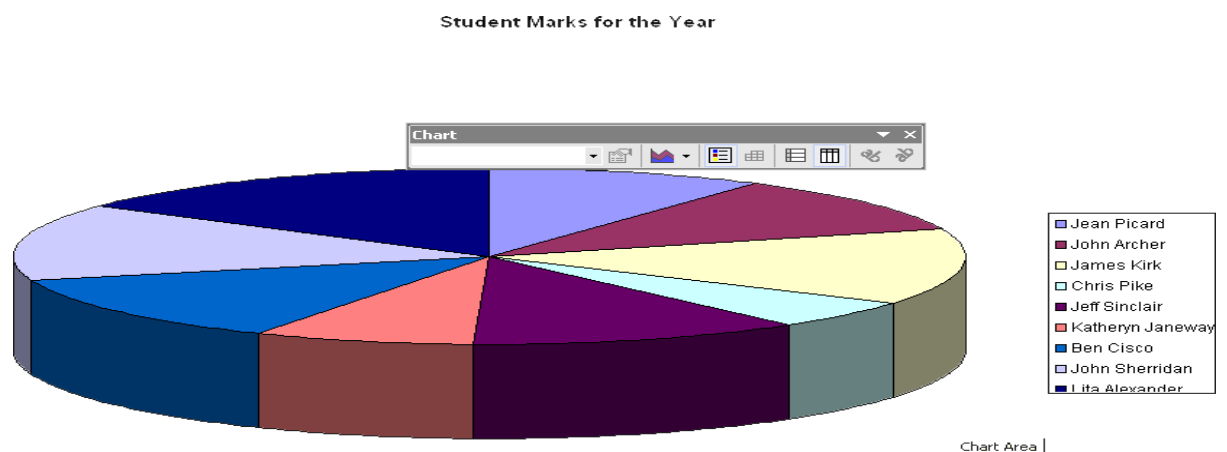
- 4) Select Pie for the Chart type and the second sub-type as shown in the example above.
- 5) Click Next to continue
- 6) Confirm that the selected cell range reads =Sheet1!\$A\$5:\$A\$14,Sheet1!\$F\$5:\$F\$14 and click Next to continue.
- 7) Change the Chart title to Student Marks for the Year and click Next to continue.



8) For the final step, select As new sheet and enter Pie Chart for the name of the sheet.

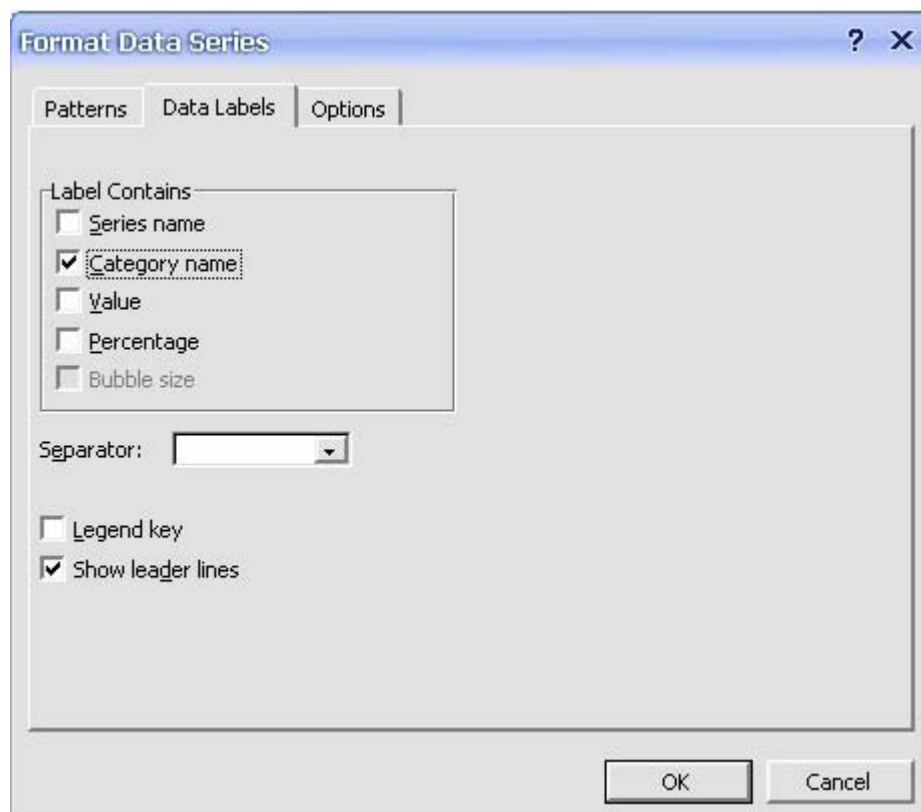
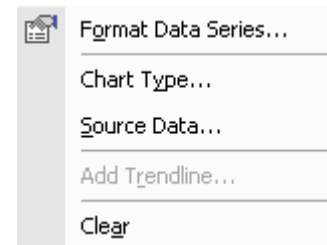


9) Click Finish to complete the chart.



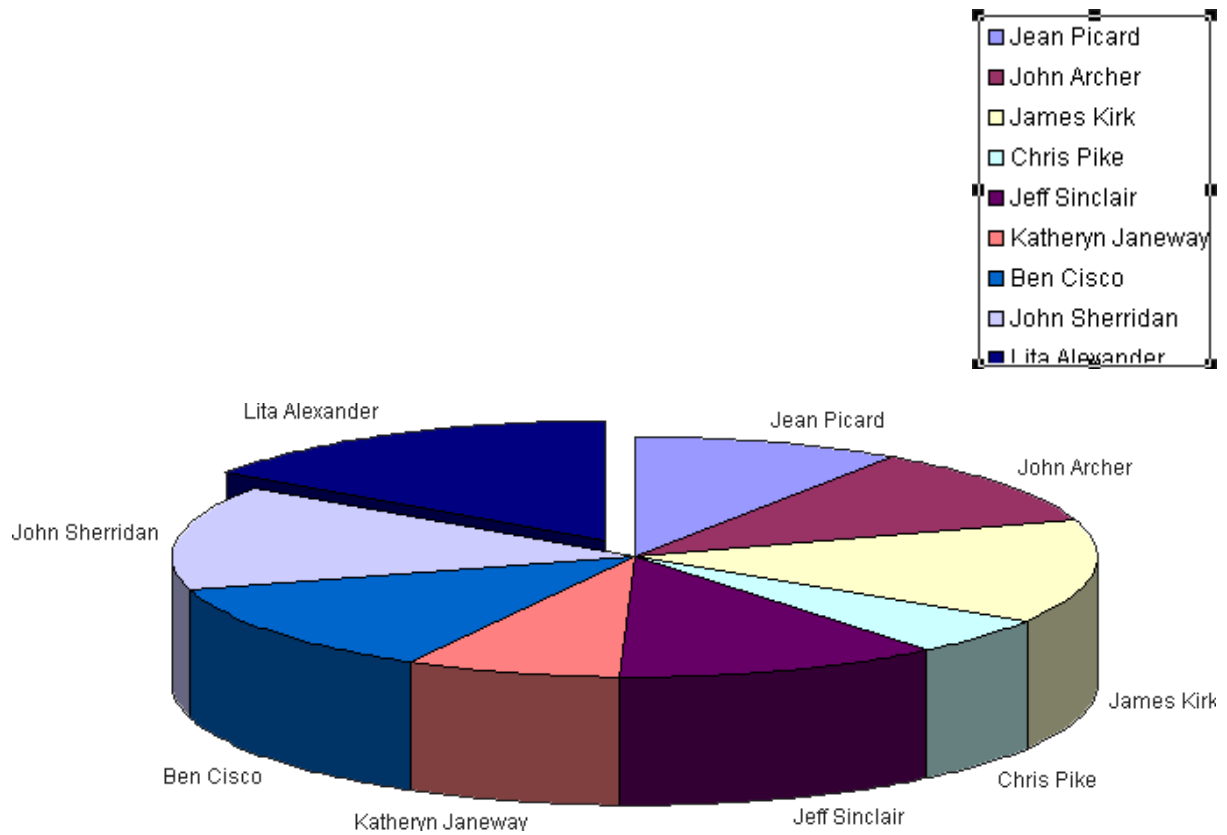
Using Microsoft Excel Creating Charts

- 10) Click on the chart to select it.
- 11) Right-click on the selected chart and select **F**ormat Data Series.
- 12) Select the Data Labels tab and select the **C**ategory Name option as shown below.



- 13) Click OK to make the change. The name of each student will now appear next to their pie slices. This means we no longer need the chart legend so we can delete it to make more room.
- 14) Click the legend to select it and press [Delete].
- 15) Click the chart to select it again and then click the pie piece for Lita Alexander. That piece will now be the only one selected.

16) Slowly drag that piece away from the centre of the pie. This technique is often used to emphasise a certain figure (such as the highest mark).



You can return to the original sheet (Sheet 1) at any time by using the sheet tabs at the bottom of the window.

LECTURE 3:

INTRODUCTION TO SOFTWARE & PROGRAMMING LANGUAGES***Software***

❖ Software is the collection of programs to perform specific task.

➤ The Software's are categorized into three types, they are:

1. System Software
2. Application Software
3. Firmware(ROM-BIOS)

1. System Software:-

- The system software is also known as the "**Operating System(OS)**" It is also known as "**Basic Software**" of PC.
- It acts as an interface between user and the system peripherals, user, application program and hardware.
- The primary goal of OS is thus to make done computer system easy to use and secondary goal is hardware resource management.
- The user interact with the computer and software through the OS.
- The OS is the first software loaded when a computer boots up.
- All computers have an OS that is used for starting the computer and running other programs (Application Programs) .
- Os perform important task like receiving input from the keyboard, mouse and sending information to the screen, keeping track of files and take appropriate action.

Ex:

Windows 95,98.
 Windows XP 2000 Professional
 Windows XP 2002 Professional (Service Pack2),
 Windows XP 2007 Professional....etc.,

Types of Operating Systems:-

- The operating system is a collection of programs used to operate a computer system.
- It takes the responsibility of computers' overall operation and supervision.
- In that way, it coordinates among various devices and programs, and provides an interface between a programmer and the computer system.
- OS is popularly used term for the abbreviation of the operating system.
- DOS became the most widely used microcomputer operating system.
- Today, DOS has been replaced on many computers by more powerful operating systems such as Windows, UNIX, Windows NT, and OS/2.
- These more powerful operating systems make it possible to run tasks on less expensive microcomputers that were once performed on expensive mainframe and minicomputers.

- Operating Systems are broadly classified, according to number of users it supports, into two categories:

- ✚ Single User Operating Systems
- ✚ Multi User Operating Systems

- Only one user can work at a time on a computer system with a single user operating system.
- More than one user can work at a time on a computer system with a multi user operating system.
 - **The following are lists of some popular operating systems:-**

Single User Operating Systems:-

- ✚ OS/2
- ✚ MS DOS
- ✚ WINDOWS 95
- ✚ WINDOWS 98
- ✚ WINDOWS XP & 2000 Versions

Multi User Operating Systems:-

- ✚ UNIX
- ✚ LINEX
- ✚ WINDOWS NT

Features of Operating System:-

- It controls overall operations of a computer.
- It coordinates among various devices and programs.
- It makes computer machines to work with users.
- It executes programs, allocates memory space, and schedules jobs.
- It provides users an interface to computing resources.
- It prevents interference between different users in a multi-user system.
- It takes users' commands and responds to them suitably.
- It deals with any faults that may occur in the computer.

2. Application Software:-

- Application software is the software designed for the specific purposes to accomplish the specific task.
- Application softwares are written by users.
- The operating system provide a set of services for the applications running on your computer, and it also provide the fundamental user interface' for your computer.

3. Firmware:-

- Firmware is one of the most common users of flash memory is for the basic input output system of your computer, commonly known as the "**BIOS**"(Bye-Ose).
- ROM-BIOS is a set of programs built in to the computer that perform the most basic, low-level and intimate control supervision operation for the computer.
- The basic purpose of ROM-BIOS is to take care of the immediate needs of the computer's hardware and to isolate all other programs from the details of how the hardware works.
- BIOS is the partly software and partly hardware.
- It is bridge between computer hardware and software

➤ **The following are some of the functions of BIOS:**

- ✚ It boots the computer
- ✚ It validates the pc's configuration.
- ✚ It provides interaction between hardware of the pc and its software.

Programming languages

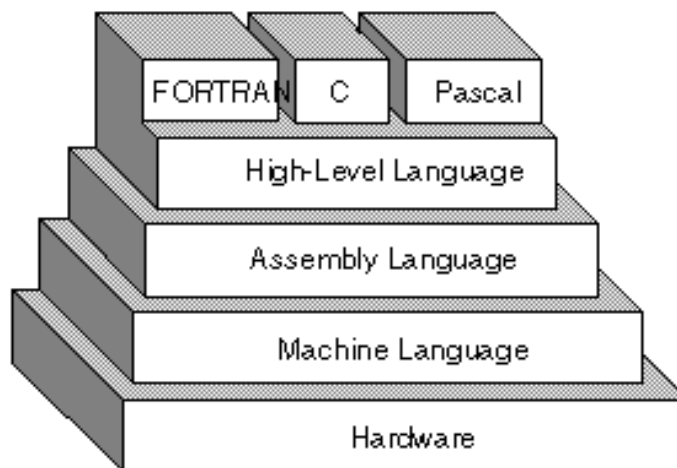
Programming Language:-

❖ **Language** is a sequential arrangement of set of *Instructions* to perform a Several tasks.

➤ Computer languages can be divided into two groups:

- ✚ High-Level Languages
- ✚ Low-Level Languages

➤ **The following figure shows the levels of the Computer Languages:**



High-Level Languages:-

- High-level languages are designed to be easier to use, more abstract, and more portable than low-level languages.
- The lowest-level programming language machine languages are the only languages understood by computers.
- While easily understood by computers, machine languages are almost impossible for humans to use because they consist entirely of numbers.
- Programmers, therefore, use either a high-level programming language or an assembly language.
- Ultimately, programs written in a high-level language must be translated into machine language by a **Compiler** or **Interpreter**.

- A vocabulary and set of grammatical rules for instructing a computer to perform specific tasks. The term **programming language** usually refers to high-level languages, such as BASIC, C, C++, COBOL, FORTRAN, Ada, and Pascal.
- Each language has a unique set of keywords (words that it understands) and a special syntax for organizing program instructions.
- The main advantage of high-level languages over low-level languages is that they are easier to read, write, and maintain.

Low-level Languages:-

- There are two different category of languages are to be contained in the lowest level languages, they are:

- i. Assembly Languages
- ii. Machine Languages

i. Assembly Languages:

- Lying between machine languages and high-level languages are languages called “**Assembly Languages**”.
- Assembly languages are similar to machine languages, but they are much easier to program in because they allow a programmer to substitute names for numbers. Machine languages consist of numbers only.
- An assembly language contains the same instructions as a machine language, but the instructions and variables have names instead of being just numbers.
- Assembly language programs are translated into machine language by a program called an “**Assembler**”.

ii. Machine Languages:-

- High-level programming languages, while simple compared to human languages, are more complex than the languages the computer actually understands, called “**Machine Languages**”.
- Each different type of CPU has its own unique machine language.
- Every CPU has its own unique machine language.
- Programs must be rewritten or recompiled, therefore, to run on different types of computers.
- A programming language that is once removed from a computer's machine language.
- Machine languages consist entirely of numbers and are almost impossible for humans to read and write.

DEFINITIONS, PROPERTIES, DIFFERENCES OF AN ALGORITHM & FLOWCHARTS

Algorithm & Flowchart

- ❖ **Algorithm** is a tool that represents program logic sequence in simple steps and in the language easily understandable by us.
- ❖ Algorithms are supported by a graphical tool called **Flowchart**.
- ❖ **Flowchart** is a tool that can help us to develop and represent graphically program logic sequence. It will also enable us to trace and detect any logical or other errors before the programs are written.
- ❖ Before you start coding a program it is necessary to plan the step by step solution to the task your program will carry out. Such a plan can be symbolically developed using a diagram. This diagram is then called a **Flowchart**.

Algorithm:-

- ❖ An algorithm is a step-by-step problem solving procedure that can be carried out by a computer.
 - **The essential properties of an algorithm are:-**
 - ✚ It should be simple.
 - ✚ It should be clear with no ambiguity.
 - ✚ It should lead to a unique solution of the problem.
 - ✚ It should involve a finite number of steps to arrive at a solution.
 - ✚ It should have the capability to handle some unexpected situations which may arise during the solution of a problem.

Advantages:-

- It is a step by step solution to a given problem which is very easy to understand.
- It has got a definite procedure which can be exacted within a set period of time.
- It is easy to first develop an algorithm, then convert it into a flowchart and then into a computer program.
- It has got a beginning and an end within which there are definite procedures to produce outputs within a specified period of time.
- It is easy to debug as every step has got its own logical sequence.
- It is independent of programming languages.

Disadvantages:-

- It is time consuming and cumbersome as an algorithm is developed first which is converted into flow chart and then into a computer program.

Flowchart:-

- ❖ Computer professionals use two types of flowcharts, Flowcharts used by computer professionals are:
 - ✚ Program Flowcharts
 - ✚ System Flowcharts

Program Flowcharts:-

- These are used by programmers.
- A program flowchart shows the program structure, logic flow and operations performed. It also forms an important part of the documentation of the system.
 - It broadly includes the following :-
 - ✚ Program Structure
 - ✚ Program Logic
 - ✚ Data Inputs at various stages
 - ✚ Data Processing
 - ✚ Computations and Calculations
 - ✚ Conditions on which decisions are based
 - ✚ Branching & Looping Sequences
 - ✚ Various Outputs

System Flowcharts:-

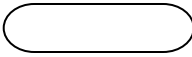
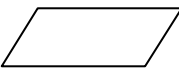
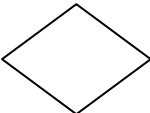
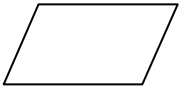
- System Flowcharts are used by system analyst to show various processes, sub systems, outputs and operations on data in a system.
 - **Use of flowcharts offers the following advantages as mentioned below:**
 - Developing the program logic and sequence.
 - A pictorial method of communicating your ideas to others.
 - The operation that will be executed at each step.
 - It shows the execution of logical steps without the syntax and language complexities of a program.
 - You can walk through your program with sample data to get the exact idea about the result of each step.
 - Compare the results obtained from the algorithm, flowchart with the expected result. Rectify any inconsistency.
 - To get logical consistency.
 - Select best option from various ways of solving the problem. This can be used to reduce the program size.
 - Simplify trouble shooting and tracing of logical errors.
 - Assist in documentation of the program.


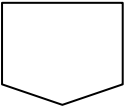
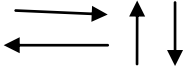
FLOWCHART SYMBOLS, ADVANTAGES & DISADVANTAGES OF FLOWCHARTS & ALGORITHMS

Flowchart Symbols

- ❖ A large number of programmers use flowcharts to assist them in the development of computer programs.
- ❖ Flowcharts has many standard symbols.
- **In using the symbols following points have to be kept in mind:**
 - All flowcharts must have a start and stop terminator.
 - All shapes must be aligned horizontally and vertically.
 - The arrows drawn must touch the shapes.
 - In case of decision shape, the yes and no branches must be clearly labeled.
 - The flowchart must have a title stating what the flowchart will do.
 - Symbols must contain the necessary information to execute the step (or) The details inside the symbol must be clearly legible.
 - In case of predefined process, the symbol must contain the name of the process.
 - The shape of the symbol is important and must not be changed.
 - The size can be changed as required.
 - The flow lines must not cross each other.

Following are various Symbols Used to Draw a Flow Charts:-

Symbol	Symbol Name	Meaning
	Terminator	A flowchart should always begin and end with this symbol.
	Process	This symbol is used to indicate a process.
	Decision	Each decision that is made is represented with this symbol. The flow comes from the top and the various possible outcomes come out of the three other "points" of the diamond.
	Input/Output	This symbol represents information entering (or) leaving a process. This is used to indicate input from the terminal(from the user)/ output to the terminal display(to the user).

Symbol	Symbol Name	Meaning
	On-Page Connector.	This symbol used to connect multiple lines of the flow chart together to other related.
	Off-Page Connector	This symbol used when you want to continue the flow chart in another page.
	Sequence (or) Direction Lines	The arrow denotes direction of the flow of the sequence of events that occur.

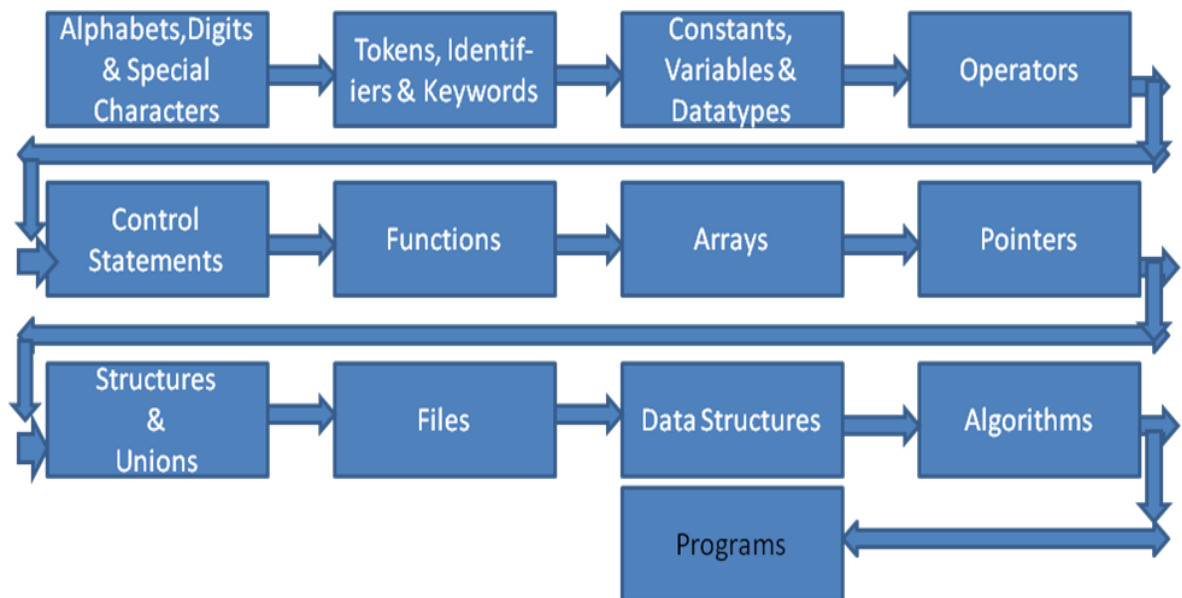
Differences Between Algorithm & Flowchart :-

S.NO	ALGORITHM	FLOWCHART
1.	An algorithm is the finite of step-by-step instructions that solve a program.	A flow chart is a diagrammatic representation or pictorial representation of the algorithm or of the plan of solution of a problem.
2.	Algorithm gives verbal representation which is almost similar to English language. Suitable for large and modular programs.	Gives pictorial representation. Suitable for small programs.
3.	Easier to understand	To understand flowchart one has to be familiar wit symbols.
4.	Drawing tools are not required	Required.
5.	Algorithm can be typed so reproduction is easy.	Flowcharts cannot be typed. So reproduction is a problem

LECTURE 4:

INTRODUCTION & FUNDAMENTALS OF 'C'***Introduction to 'C'***

- ❖ C is a general purpose structured programming language developed at Bell laboratories in 1972.
 - ❖ It was designed and written by a system programmer named Dennis Ritchie.
 - ❖ C was powerful programming language, which is attracting considerable attention worldwide because it is reliable, simply, easy to use, and highly portable(i.e., a program written in C can be transferred easily from one computer to another computer).
- **Applications of C:-**
- ❑ C language is used for developing database systems, graphics packages, spread sheets, CAD/CAM applications, word processors, office automation, scientific and engineering applications.
- Learning C is easier than any other programming language.

The following are steps in learning C:-

STEPS IN LEARNING 'C' (CHARACTER SET, IDENTIFIERS, KEYWORDS)

Steps in Learning 'C'

Character Set:-

- ❖ A character set denotes an alphabet, digit, (or) special characters can be combined to form variables.
- ❖ C uses constants, variables, operators, keywords, and expressions as building block to form a basic C program.
- Valid characters used in C programming are as follows:

Alphabets:

Lower case: a b c x y z

Upper case: A B C X Y Z

Digits:

0 1 2 8 9 10

Special Characters:

+ plus sign	– minus sign	_ underscore
/ slash	! exclamation mark	# number sign
< less than	> greater than	= equal to
? question mark	% percent sign	' single quotation
" double quotation	; semicolon	, comma
{ left flower brace	} right flower brace	[left bracket
] right bracket	(left parenthesis) right parenthesis
& ampersand	vertical bar	* asterisk
. dot operator	back slash	

Tokens:-

- ❖ Tokens are the smallest individual unit in a program.

➤ **Tokens supported in C are as follows:**

- Identifiers
- Keywords
- Constants
- String literals
- Operators

Identifier:-

- A C program is usually written with these tokens, as per syntax of the C language.
- An identifier is a string of alphanumeric characters that begins with an alphabetic character (or) an under score character that are used to identify (or) name various programming elements such as:

- ❑ variables, function names, array names, tags and members of structures, unions, typedef names , enumeration constants and so on...
- An underscore character is considered as a letter in identifiers and it is usually used in the middle of an identifier.

Rules for constructing identifiers:-

- ❑ The first character in an identifier must be an alphabet (or) an underscore and can be followed by any alphabets (or) digits or under scores.
- ❑ No commas (or) blank spaces are allowed within an identifier.
- ❑ No special character other than the underscore can be used to represent identifiers.
- ❑ The name used to represent an identifier should not be a keyword.
- ❑ Identifiers are case sensitive.
- ❑ Identifiers can be of any length, but first 31 characters are recognized.

➤ Valid examples of identifiers:-

Chennai PRIYA FLOAT average_value SuChiTra
KUMARA _raja One_And_Only mark_1

➤ Invalid examples of identifiers:-

786 (sum) 100th cost\$ [Rupa] FIVE STAR
"Vanitha" float

Keywords:-

- Keywords are also referred, as "**reserved words(predefined words)**" are words whose meaning has been already defined in C compiler.
- All keywords have fixed meanings and their meanings cannot be changed.
- Keywords are used for their intended purpose and they cannot be used as identifiers.
- All keywords must be compulsorily written in lowercase.

➤ The followings are keywords used in C:-

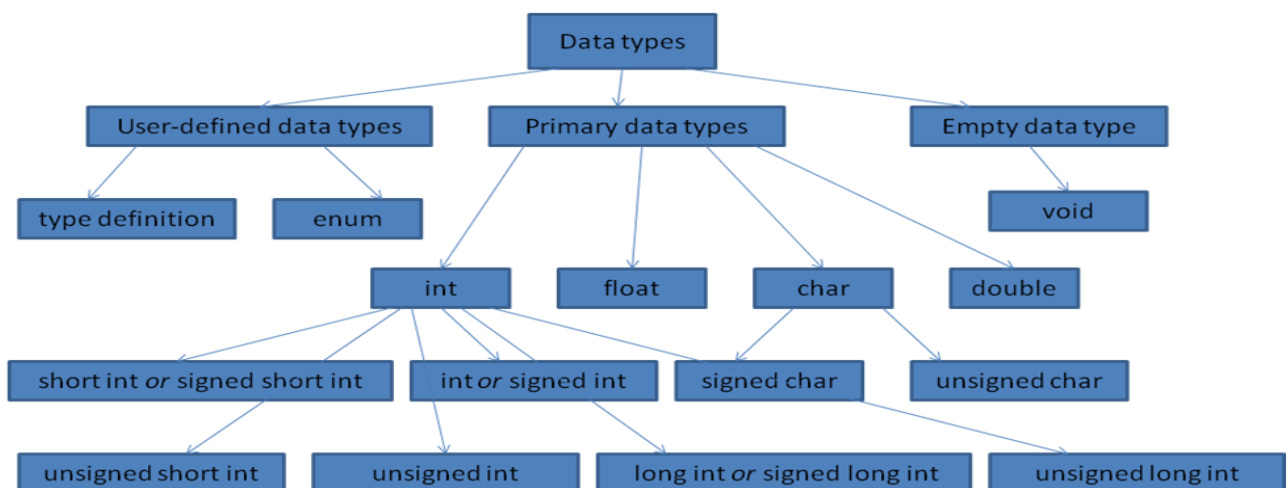
auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

STEPS IN LEARNING 'C' (DATA TYPES)***Steps in Learning 'C'*****Data Types:-**

- ❖ The data types supported in a language dictates the type of values, which can be processed by the programming language.
- ❖ C language is rich in its data types.
- ❖ It supported wide variety of data types each of which may be represented differently with in computer's memory, which allows the programmer to select the appropriate type to the needs of the application.

➤ Data types in C can be broadly classified as three types, they are

1. Primary data types
2. User-defined data types
3. Empty data type.



- All C compilers supports four primary data types namely,
- Character data type represented as **char**
 - Integer data type represented as **int**
 - Floating-point data type represented as **float**
 - Double-precision floating-point data type represented as **double**.

1. Primary Data types:-

Data type	Size (bytes)	Range
char	1	-128 to 127
int	2	-32768 to 32767
float	4	3.4e-38 to 3.4e+38
double	8	1.7e-308 to 1.76+308
long double	10	3.4 e-4932 to 1.1e 4932

Integer Data Types:-

- C deals with several kinds of numbers. Whole numbers, which are used frequently, are referred as **integers**.
- It occupies two bytes for their storage.
- In order to have control over the range of numbers and storage space in memory short int, int, long int in both signed and unsigned forms are used.
- The long int data type represents large integer values and requires double the amount of storage as regular integer data type.
- The unsigned long int data type represents large integer values that are used to store large range of values than long int data type.

Data Type	Size(bytes)	Range
short int (or) signed short int	2	-32,768 to 32,767
int (or) signed int	2	32,768 to 32,767
unsigned short int	2	0 to 65,535
unsigned int	2	0 to 65,535
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 Data Types:-

- C deals several kinds of numbers.
- Real numbers, which are used frequently, are called as floating-point numbers.
- The float data type is used to store a floating point number.
- Floating point data types store numerical values with s fractional portion.
- They usually occupy 4 bytes of memory for their storage. These are known as "**single precision numbers**".
- When the range provided by a float data type is not sufficient, double data type is used.
- The double data type represents the same data type, that float represents but with a greater precision hence it requires 8 bytes of memory for their storage in the memory. These are known as "**double precision numbers**".
- To increase the precision further we can use long double which uses 10 bytes of memory for their storage. These are known as "**extended precision numbers**".

Character Data types:-

- Character data type is used to represents individual characters.
- The char data type generally requires one byte of memory for their storage.
- In order to have control over the range of characters and storage space in memory, signed and unsigned char data type are used.

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

2. User Defined Data types:-

- C supports a feature known as type definition that allows the user to define new data types that are equivalent to existing data types.
- The user-defined data type identifier can later be used to declare variables.
- There are two types of user defined data types are used in C:

- ❑ **Type Definition**

- ❑ **Enumerated Data Type**

Type Definition:-

- The keyword **typedef** is used to define new data type names.
- Note that you are not actually creating a new data type, but rather defining a new name for an existing data type.
- The general form (or) syntax of the defining a new data type is:

typedef data type identifier;

Where

- ❑ identifier refers to new name(s) given to the data type.

- They can later used to declare variables as

```
age child, adult;
average mark1,mark2;
string name[20];
```

Where

- ❑ child and adult are declared as integer variables/identifiers,
- ❑ mark1 and mark2 are declared as floating-point variables/identifiers,
- ❑ name is declared as character array variable/identifier.

- **typedef** can also be used to define structures.
- The main advantage of type definition is that, you can create meaningful data type names for increasing the readability of the program.
- They also suggest the purpose of the data type names used in the program.

Examples of “type definition” are:

```
typedef int age;
typedef float average;
typedef char string;
```

Where

- age symbolize int,
- average symbolize float and
- string symbolize char.

Enumerated Data Type:-

- The enumerated data type gives an opportunity to invent your own data type with a set of named integer constants represented by identifiers that specifies all valid values a variables of that type may have.
- They can be used in any expression where integer constants are valid.
- The general form (or) syntax of Enumerated Data Type is:

enum identifier {enumeration _list} list_of_variables;

Where

- enum is a keyword, which is used to specify the enumerated data type which can be used to declare the enumeration _list also referred as **enumeration constants**.

Ex:

```
enum month{JAN,FEB,.. . . . . . NOV,DEC};
```

Where

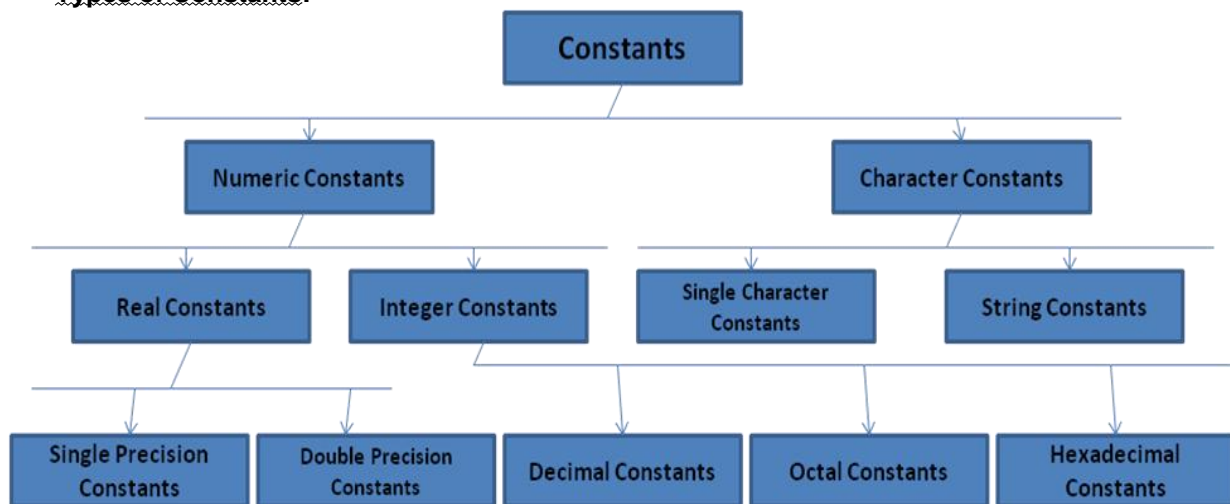
- New data type month in which the enumeration _list is JAN,...,DEC.
- The values for the enumeration _list are set automatically to integers 0 to 11.

3. Empty Data Types:-

- The keyword **void** is used to indicate an empty data type.
- This data type is not used declare any variables, but used to indicate nothing.

STEPS IN LEARNING 'C' (CONSTANTS)***Steps in Learning 'C'*****Constants:-**

- ❖ Constants are tokens in C representing fixed numeric (or) character values that do not change during execution of a program.

Types of Constants:-

- The Constants are broadly divided into two types:

1. Numeric Constants
2. Character Constants

1. Numeric Constants:-**Integer Constants:-**

- Integer constant refers to a sequence of digits without a decimal point. C deals several kinds of numbers, one of the most frequently used is the whole numbers, usually called as an "***integer***".
- An integer constant refers to a sequence of digits without a decimal point.
- An integer constant preceded by a unary minus may be considered to represent a negative constant.

Ex: -43

- Integer constants can be specified in three ways. They are,

- Decimal integer constant
- Octal integer constant
- Hexadecimal integer constant

Ex:

- The value 35, can be written in any of the three ways as,

- ✚ 35 representing a decimal constant

- ✚ 043 representing a octal constant

- ✚ 0x23 representing a Hexadecimal constant

- The leading 0(zero) to an integer constant specifies, that it is an octal integer constant.
- It can containing digits from **0** to **7**.
- The leading **0x** (or) **0X** to an integer constant specifies, that it is a hexadecimal constant.
- It can contain digits from **0** to **9** and letters **a** to **f** (or) **A** to **F**.

- **Rules for constructing integer constants:-**

- ✚ An integer constant may be positive or negative(default sign is positive).
 - ✚ An integer constant should not have a decimal point.
 - ✚ No commas and blank spaces are allowed in an integer constant.
 - ✚ An integer constant, should not use exponent **e**.

Real Constants:-

- Real Constants are often called as floating-point constants.
 - They can be written in two forms. They are:
 - Fractional form
 - Exponential form

Fractional form:-

- ❖ Fractional part consists of a series of digits representing the whole part of the numbers, followed by a decimal point, followed by a series of digits representing the fractional part.

- **Rules for constructing real constants(fractional form):-**

- ✚ A real constant may be positive (or) negative (default sign is positive).
 - ✚ A real constant must have a decimal point.
 - ✚ No commas and blank spaces are allowed in a real constant.

Exponential Form:-

- Exponential form of representation of real constant is usually used if the value of a constant is either too small (or) too large.

- In this representation, the real constant is represented in two parts.
 - The part appearing before e is called **mantissa**, where as the part following e is called **exponent**.
 - The exponent consists of an optional plus (or) minus sign followed by a series of digits.

Ex:

343×10^{-6} is represented in exponential form as 343e-6.

➤ **Rules for constructing real constants(Exponential form):-**

- ✚ The mantissa part and the exponential part should be separated by a letter **e**.
- ✚ The mantissa part and the exponential part may be positive (or) negative(default sign is positive)
- ✚ The mantissa and exponent part must have at least one digit.
- ✚ No commas and blank spaces are allowed in a real constant.

2. Character Constant:-

Single Character Constants:-

- A Single Character Constant (or) Character Constant contains a single character enclosed within a pair of single quotes.
- A Single Character Constant must be either a single alphabet (or) a single special character enclosed in single quotes.
- The length of the Single Character Constant is only one.

Ex: 'A' 'k' '4' '*' '_'

String Constant:-

- A character string, a string constant (or) literal consists a sequence of characters enclosed within in a double quotes.
- A string constant may consists of any combination of digits, letters, escape sequences, and spaces.
- The end of the string is marked with a single character called as the NULL character '\0'.
- It occupies two bytes of size for their storage capacity.

Ex: "Kumar" "Food Science" "434" etc....;

➤ **Rules for Constructing string constants:-**

- ✚ A string constant may be a single alphabet (or) a digit, (or) a special character (or) sequence of alphabets (or) digits enclosed in double quotes.
- ✚ Every string constant ends up with a NULL character, which is automatically assigned.

STEPS IN LEARNING 'C' (VARIABLES, ESCAPE SEQUENCES)

Steps in Learning 'C'

Variables:-

- ❖ A quantity which may vary during program execution, is as a variable.
- Variable name is a arbitrary name, an identifier used to refer to the area of memory in which a particular value is stored.
- These memory locations can contain an integer, real, single character, (or) a string constant.

Declaration of Variables:-

- A data item must be assigned to a variable at some point in the program.
- The information represented by the variable can change during execution of the program, but data type associated with the variable cannot change.
- A variable name used by the programmer can be in a meaningful way to reflect its use (or) function (or) nature in the program.
- All variables used in the program must be declared before all executable statements(i.e.; at the start of each function after the opening flower braces ({) of the program.
- Declaration of variable begins with its type followed by one (or) more variable names.
- Declaration assigns the variable to a specific data type and there by allocate memory to the variable depending upon the data type.

Ex:

```
int a1,a2,a3;
char c1,c2,c3;
float x1;x2;x3;
```

- In the first declaration, a1,a2,a3 are variables declared as type integer.
- In the next declaration c1,c2,c3 are variables declared as type character.
- In the next declaration x1,x2,x3 are variables declared as type floating-point variables.

Initialization of Variables:-

- Variables can also be initialized when they are declared.
- This is done by adding an equal sign followed by the required value to be initialized after the declaration.

Ex:

```
int a=6, b=8;
char c1="kumar";
float x1=434.4;
```

- ✚ The first declaration declares two variables a and b of type integer and their by initializing the variables with the values 6 and 8 respectively.

Types of variable:-

Constant Variables:-

- The keyword **const** is used for this purpose.
- It protects variables from modification during execution of a program.

Ex:

```
const int x= 34;
const float y=234;
```

- In the first declaration, x is a variable declared as type integer.
- In the next declaration, y is a variable declared as type floating-point.
- The keyword **const** is used for this purpose.
- It protects variables from modification during execution of a program.

Volatile Variables:-

- The variables that can be changed at any time by a function in which it is present (or) by any other function in the same program are called as **volatile variables**.
- The volatile is used for this purpose.

Ex: volatile int z=5000;

- If the value of a variable in the current function is to be maintained constant and desired not to be changed by any other external function, then the variable should be declared as:

```
volatile const int z=5000;
```

Escape Sequences:-

- Certain constants are non printable, which means that they are not displayed on the screen (or) in the printer, called as escape sequences.
- C supports certain escape sequences(backslash character constants) that are used in output statements.
- Although they consist of two characters, they represent only one character.

Ex:

\0 -- null **\b** – blank space **\t** – horizontal tab **\n** – new line

LECTURE 5:

STEPS IN LEARNING 'C' (OPERATORS)***Steps in Learning 'C'*****Operators:-**

- ❖ An Operator is a symbol (or) a special character that tells the computer to perform certain mathematical (or) logical operations, which is applied to operands to give a result.
 - Operators are used in programs to manipulate data and variables.
 - The data items that operators act upon to evaluate expressions are called as **Operands**.
 - The Operators are used in C are:
 1. Arithmetic Operators
 2. Unary Operators
&
Increment and Decrement Operators
 3. Relational Operators
 4. Equality Operators
 6. Logical Operators
 7. Bitwise Operators
 8. Conditional Operator
 9. Assignment Operators
 10. Special Operators

1. Arithmetic Operators:-

- ❖ C language has all the basic arithmetic operators.
- following table shows the various arithmetic operators and their meaning:

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo operation(remainder after integer division)

2. Unary Operators:-

- ❖ The operator that act upon a single operand to produce a new value are called as “**unary operators**”.
- Unary operators are written after their operands.
- The following table shows the various unary operators and their meaning:

Operator	Meaning
-	Unary minus
++(increment operator)	Increment by 1
-- (decrement operator)	Decrement by 1
sizeof	Returns the size of the operand.

Unary minus:-

- The most common unary operation is the unary minus, which is distinctively different from the arithmetic operator, which denotes subtraction.
- The subtraction operation requires two operands, where as the unary operation, with unary minus requires only one operand.
- The unary minus is used to indicate (or) change the algebraic sign of a value.

sizeof operator:-

- The **sizeof** operator is not a library function but a keyword, which returns the size of the operand in bytes.
- The sizeof operand always precedes its operand.
- The following are various expressions and results of sizeof operator:

Expression	Result
sizeof(char)	1
sizeof(int)	2
sizeof(float)	4
sizeof(double)	8

3. Relational Operators:-

- ❖ Relational operators are symbols that are used to test the relationship b/w two variables (or) b/w variables and constant.
- This operators are used for checking conditions in control statements.
- The following table shows the various relational operators and their meaning:

Operator	Meaning
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to

4. Equality Operator:-

- Closely associated with the relational operators are two equality operators.
- The following table shows the equality operators and their meaning:

Operator	Meaning
==	equal to
!=	not equal to

5. Logical Operator:-

- ❖ In addition to arithmetic and relational operators, C has three logical operators for combining logical values and creating new logical values.
- The following table shows the various logical operators and their meaning:

Operator	Meaning
!	NOT
&&	Logical AND
	Logical OR

- The result of the **logical AND** operation will be true, if the both operands are true.
- The result of the **logical OR** operation will be true if either of the operand is true or both the operands are true.
- The **logical NOT** operator simply reverses the value of the expression that follows it(i.e; true becomes false and false becomes true).
- **Ex:** The various expressions when x=9,y=5,z='a' and their results are:

Expression	Result	Value
(x>=5) && (z=='a')	True	1
(x>=6) (z=='a')	True	1
(x>=15) (z=='b')	False	0
(x>8)	True	1
!(x>8)	False	0

6. Bitwise Operator:-

- ❖ The bitwise operators are the bit manipulation operators.
- They can manipulate individual bits within the piece of data.
- These operators can operate on integers and characters but not on floating point numbers having double data types.
- The following table shows the various bitwise operators and their meaning:

Operator	Meaning
~	bitwise complement
&	bitwise-AND
	bitwise-Or
^	bitwise-exclusive-OR
<<	bitwise shift left operator
>>	bitwise shift right operator

7. Conditional Operator:-

- ❖ An expression that evaluates conditions is called as **conditional expressions**.
- The operator used to evaluate conditional expression is called as a **conditional operator** denoted by (**?:**).
- It is also referred to as "**ternary operator**", since it takes three operands as its arguments.
 - The general form or the syntax of the conditional operator is:

expr1 ? expr2 : expr3;

- ❖ When evaluating a conditional expression, expr1 is evaluated first.
 - If expr1 is true,expr2 is alone evaluated.
 - If expr1 is false,expr3 is alone evaluated.

Ex:

a = ((b < 2) ? 8 : 6);

- If b is lesser than 2, the value of a will be 8.
- If b is grater than 2, then the value of a will be 6.

8. Assignment Operators:-

- ❖ Assignment operators are used to assign the result of an expression to a variable.
 - The **equal(=)** is an assignment operator.
 - It is used to form a assignment expressions, which assigns the value of an expression to an identifier, which may be written in the form of

Identifier=expression;

Where

- identifier generally represents a variable and expression represents a constant (or) a variable (or) complex expression.
- In addition, C has a set of shorthand assignment operators of the form:

Identifier<operator>=expression;

- The following table shows the various assignment operators and their meaning:

Operator	Meaning
+=	Assign sum
-=	Assign sub
*=	Assign product
/=	Assign quotient
%=	Assign reminder

9. Special Operators:-

- ❖ Some of the special operators used in C are listed below.
 - These operators are referred as separators.
 - ❑ Ampersand(&)
 - ❑ Comma(,)
 - ❑ Asterisk(*)
 - ❑ Hash(#)
 - ❑ Braces({ })
 - ❑ Parenthesis(())
 - ❑ Colon(:) and Semicolon(;) etc.,

STEPS IN LEARNING 'C' (STATEMENTS)

Steps in Learning 'C'

Statements:-

- ❖ A statements in a program carries out some specific action.
 - Statements used in C are classified as three types, they are
 1. Expression Statements
 2. Compound Statements
 3. Conditional Statements

1. Expression Statements:-

- An expression is a combination of constants, variables, operators, and function calls written in any form as per syntax of the C language.
- The value produced by these expression can be stored in variables and used as apart for evaluating larger expressions.
- An expression statement consists of valid C expressions, ending with a semicolon.

Variable = expression;

- The expression refers to a constant value (or) an arithmetic expression (or) a function which assigns a value to the variable.

Ex:

```
age=21;
result=pow(2,4);
area_of_circle=3.14*r*r;
```

2. Compound Statements:-

- A group of valid C expression statements placed within an opening flower brace '{' and a closing flower brace '}' is referred as a "***compound statement***".

Ex:

```
{
    x=(a+(b-c)*c);
    y=(a-b+d);
    z=(a*b-c)/d;
}
```

- A compound statement can also have another compound statement, (or) a control statement in it.

3.

4. Control Statements:-

- The statements, that we have seen so far, are normally executed as they appear in the program.
- By default, the statements in the program are executed sequentially.
- However, in practice we have situations where we have to change the order of execution of statements until the some specified conditions are met.
- The control statements alter the execution of statements depending upon the conditions specified inside the parenthesis.

Ex:

```
    if(a>=b)
    {
        .....
        Expression Statement(s);
        .....
    }
```

STEPS IN LEARNING 'C' (HEADER FILES, INPUT & OUTPUT FUNCTONS) FORMATTED & UNFORMATTED I/O FUNCTIONS)

Steps in Learning 'C'

Header Files:-

- ❖ Library functions that have similar functions are grouped together as programs (with file extensions .h) in separate library files.
- ❖ These library files are supplied as a part of the C compiler.
- ❖ In order to use library function it is necessary to include a specific information within the main portion of the program, called the header files.
- The general form (or) syntax of including a header file in the program is

#include < filename.h >

Where

- ❑ filename represents the header file which has their file extension as .h.

- The different header files in C language are:

stdio.h conio.h math.h string.h

Input & Output in C

- ❖ One of the primary advantages of modern computers is their ability to communicate with the user during program execution.
- ❖ This feature enables the programmer to enter the values for the variables, when the program is in execution without altering the program.
- ❖ Once the required value has been entered and the ENTER key is pressed, program execution starts again, and the output is displayed in the screen.
- ❖ To perform input and output operations, C has a number of input and output functions.
 - These functions can be broadly classified into two types. They are:

- ❑ Formatted I/O functions
- ❑ Unformatted I/O functions

Formatted I/O functions:-

- The formatted I/O functions read any type of data as input from the user and displays of data output in the screen.
- They use format strings (conversion specifiers) for identifying the type of data involved.
 - ❖ The scanf() function is formatted input function
 - ❖ The printf() function is formatted output function

Unformatted I/O functions:-

- All unformatted I/O function reads and displays only char data type.
- They don't use format strings for identifying the type of the data involved, since all data are considered in character format.
- The `getch()` , `getche()` , and `getchar()` are single character unformatted input functions.
- The `put()` , `putchar()` , are single unformatted input functions.
- The `gets()` , `puts()` , are unformatted I/O functions.

LECTURE 7:

BASIC STRUCTURE OF A SIMPLE 'C' PROGRAM***Structure of 'C' Program***

- ❖ These header files contains information about various library functions that are functionally same.
- ❖ Every C program contains a number of building blocks.
- ❖ These building blocks should be written in a correct order and procedure, for the C program to execute without any errors.
- To know about the building blocks in detail, let us see a sample program for calculating the sum of two integers and to display its result.

```

/* Program to calculate the sum of two integers */      /* Title comment */
#include<stdio.h>                                     /* Library file Access */
void main( )                                         /* Function*/
{
    int x, y, sum;                                  /* Declaration of variables*/
    printf(" Enter the values of x and y:\n");      /*Output statement*/
    scanf("%d %d", &x, &y);                          /*Input statement*/
    sum= x+y;                                       /*Assignment statement*/
    printf("The sum of the variables x and y = %d",sum);/*Output statement*/
}

```

- ❖ Comments are added at each end of the statement in order to understand the overall program organization easily. Let us see each comment statement in detail.

/* Title Comment*/

- Any comment statement used in the program must be enclosed within /* */.
- The comment statement is used to identify the purpose of the program.
- This statement is usually specified at start of a program, but they can be specified any where in the program.
- This may (or) may not be present in the program as per the programmer wish.
- The comment statement is not a executable statement.
- The compiler skips the line when it encounters a comment statement.
- Comment statements may be included in between the program to understand each statement used in the program.
- It's a good programming practice to include comment statements in a program.

/* Library file Access */

- The second line in the program #include<stdio.h> refers to a special file which contains, the information about the various input and output operations which must be included in the program when it is compiled.

- The header file `stdio.h` contains the information of all the standard input and output functions in C.
- There are many header files like `math.h` , `string.h`, etc., which are used in the programs depending upon the library functions used in the program.
- Header files are usually enclosed in `< >` (or) “ ”.
- If u do not include the header file `stdio.h` in the program, the C compiler may complain about undefined functions and data types used in the program.

/* Function*/

- The third line in the program is `main()` is a function, which may consists of one (or) more statements.
- Every C program must contain at least one function which must be a `main()`.
- A C program will always begin by executing the `main()` function.
- This `main()` function will access other functions used in the program.
- The empty parenthesis followed by `main()` indicates that this function does not include any arguments.
- The remaining lines of the program are enclosed within a pair of braces.
- The statements enclosed within a pair of braces are referred as the body of the functions.
- These lines are within the compound statement `main()`.

/* Declaration of variables*/

- A declaration of a variable is a statement that gives information to the C compiler about the variables to be used in the program.
- General form (or) the syntax of declaring a variable is

Data type variable(s);

- The declaration consists of a data type, followed by one (or) more variable names, separated by commas ending with a semicolon.

Ex:

`int x, y, sum;`

Where

- `int` is the data type in C and `x` and `y` are variables and `sum` is variable names to be used in the program.
- The type of variable informs the compiler how the variable values are to be stored(i.e., how much of memory space is to be allocated).
- Note that all variables used in the program must be declared before all executable statements.

/*Output statement*/

- The other lines in the program are statements.
- The first statement is the printf() statement, that generates a request to the user to give information (i.e., values) for the variables x and y for summation.
- The printf() statement is usually called as the output statement since it display the message on the screen.
- The information to be displayed in the screen must be specified within double quotes(“ ”)

/*Input statement*/

- The value for the variables x and y are entered into the computer through the scanf() statement, which is below the printf() statement.
- The scanf() statement is usually called as the input statement because it reads the values to be processed by the computer.

/*Assignment statement*/

- The fourth line is an important for this program since it adds the two valued of x and y entered through the keyboard.
- This statement is called as an assignment statement sine it assign the value of x+y to the variable name sum.

/*Output statement*/

- The last line is the printf() statement ,which displays the output of the program (.e., the summation of x and y) in the screen.

➤ **Important points to be noted while writing a ‘C’ program:**

- All statements must be written in lowercase letters.
- However, symbolic constants , identifiers can be written in lowercase and/or upper case letters.
- All statements must be end with a semicolon.
- Comment statements can be included wherever necessary in the program . It’s a good programming practice to include comment statements in a program.
- White spaces during variable declaration , within arguments in functions, before and after operators.
- The opening and closing flower braces used in a program should be balanced, i.e.; if opening braces are three, then the closing braces should also be three.

The main() Function

- ❖ A function is a self-contained program segment that carries out some specific, well defined task whenever it is accessed.
- ❖ A program can have one or more functions but there should be at least one function, which must be a main() function.
- ❖ Empty parenthesis following the word **main** are necessary.
- ❖ Some programmers place the main() function at the beginning of the file, and other use it at the end. Regardless of their location, every C program must contain a main() function.
- ❖ The main() function consists of two parts namely, the **declaration part** and the **execution part**.
- ❖ The declaration part declares all the variables used in the executable part.
- ❖ Initialization of variables can also be done in the declaration part.
- ❖ Declaration of variables allocates a memory spaces according to their data types specified.
- ❖ The execution part contains one (or) more statements following the declaration part.
- ❖ The execution part uses all the variables declared in the declaration part.
- ❖ The declaration and execution parts must appear between the opening and the closing braces of the main() function.
- ❖ The program begins its execution at the opening flower brace '{' and ends its execution in the closing flower brace '}'.
- ❖ The closing flower brace of the main() function is the logical end of the program.
- ❖ All statements in the declaration and executable part ends with semicolon.
- ❖ A main() function may (or) may not contain a declaration statement but it should have an executable statement.
- ❖ The empty parenthesis followed by main() indicates that this function does not include any arguments.

LECTURE 8:

DECISION MAKING (OR) CONTROL STATEMENTS***Control statements***

- The statements, that we have seen so far, are normally executed as they appear in the program.
- By default, the statements in the program are executed sequentially.
- However, in practice we have situations where we have to change the order of execution of statements until the some specified conditions are met.
- The control statements alter the execution of statements depending upon the conditions specified inside the parenthesis.
 - The different decision control (or) decision-making statements in C are:
 - The if statement
 - The if-else statement
 - The multiple if-else statements
 - The nested if-else statements
 - The switch statement

The if statement:-

- Like many programming languages, C also uses the keyword if to implement decision control statements.
- **The general form (or) syntax of an if statement is:**

```

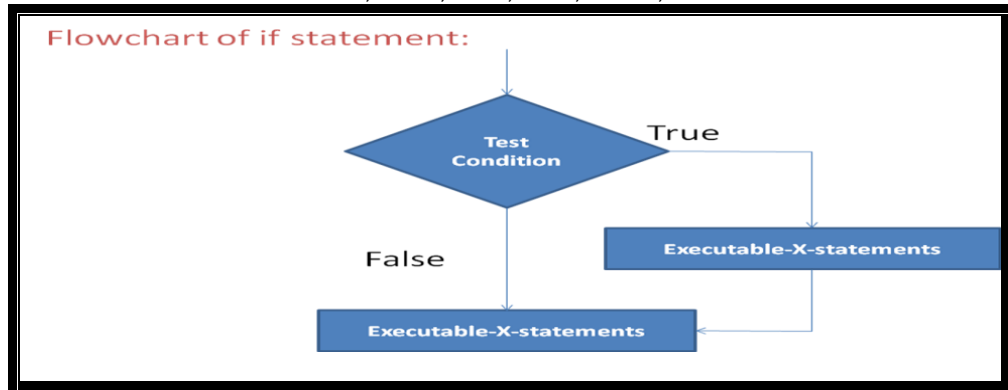
if(condition)
{
    executable -X-statement(s);
}
executable -Y-statement(s);
```

- The above syntax shows the operation of an if statement.
- The condition used in an if statement should be specified within parenthesis.
- When a program with if statement is evaluated the condition specified after the if statement is executed first.
- If the condition is true then the executable-X-statement(s) within the flower braces will be executed.
- If the condition is false, the control will be directly transferred to the statement outside the flower braces and executes executable-Y-statement(s) without executing executable-X-statement(s).
- The executable-X-statement(s) referred as the body of the statement, should enclosed within flower braces.

Ex:

If a and b are two variables the actual conditions are:

$a==b$, $a!=b$, $a>b$, $a<b$, $a>=b$, $a<=b$

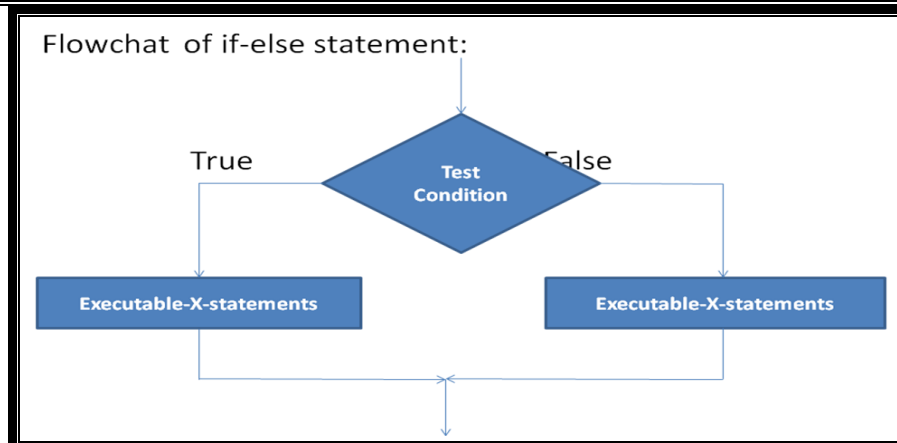


The if-else statements:-

- The if-else statement is an extension of an ordinary if statement.
 - The if statement by default will execute a single statement (or) group of statements when the condition is true, but if does nothing when the condition is false.
 - Hence, we are using an else statement to execute a single statement (or) group of statements when the when the condition is false.
- **The general form (or) syntax of an if-else statement is:**

```

if(condition)
{
    executable -X-statement(s);
}
else
{
    executable -Y-statement(s);
}
  
```



- The above syntax shows the operation of an if statement.
- The condition used in the if statement should be specified within parenthesis.
- If condition is true the statement(s) following if until else will be executed.

- If the condition is false the statement(s) following the else will be executed.
- The group of statements after the if up to and not including the else are called as the “**if block**”.
- The group of statements after the else are called as the “**else block**”.

The Multiple if else statement:-

- When a series of decisions are involved we have to use more than one if-else statement called as “**multiple if’s**”.
 - Multiple if else statements are much faster than a series of if-else statements, since the if structure is exited when any one of the condition is satisfied.
- **The general form (or) syntax of an Multiple if-else statement is:**

```

if(condition)
{
    executable -X-statement(s);
}
else if(condition)
{
    executable -Y-statement(s);
}
else
{
    executable -Z-statement(s);
}

```

The Nested if-else statements:-

- One (or) more if-else statements inside an if-else statement is called as “**nested if statements**”.
- **The general form (or) syntax of an Multiple if-else statement is:**

```

if(condition)
{
    if(condition)
    {
        executable -X-statement(s);
    }
    else
    {
        executable -Y-statement(s);
    }
    else
    {
        executable -Z-statement(s);
    }
}

```

The switch statement:-

- Like an if statement the switch allows us to make a decision from a number of choices. It is usually called as a “**switch-case statement**”.
- The switch statement causes a particular group of statements to be chosen from several available groups.
- The selection is based upon the current value of an expression, which is included within the switch statement.

➤ **The general form (or) the syntax of the switch statement is:**

```

switch(expression)
{
    case constant 1:
        statement(s);
        break;
    case constant 2:
        statement(s);
        break;
    ...
    case constant n:
        statement(s);
        break;
    default :
        statement(s);
        break;
}

```

- Each case is labeled by one (or) more integer valued expressions.
- If a case matches the expression value, execution starts at that case, and ends when it encounters a break statement all case expressions must be different.
- The case labeled default is executed if none of the other cases are satisfied.
- A default is optional in switch statement. If a default statement is not present and if none of the other cases match, no action takes place.
- The default group may appear anywhere within the switch statement, it need not be placed at the end.

LECTURE 9:

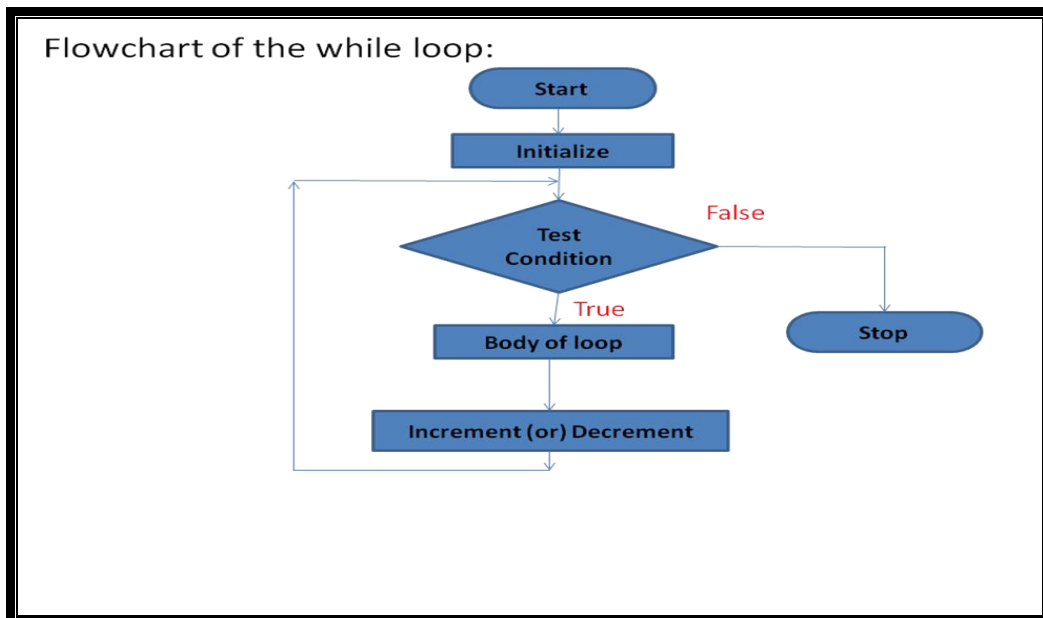
BRANCHING & CONCEPT OF LOOPING STATEMENTS***Branching & Looping***

- One of several possible actions will be carried out depending upon the outcome of the logical test is referred as branching.
- Some portion of the program (or) segment of the program has to be executed several number times (or) until a particular condition is being met. This is referred as looping.
- Some portion of the program has to be specified several number of times (or) until a particular condition is satisfied.
- Such repetitive operation is done through a loop structure.
- The loop concept is essential and it is fundamental for good programming.
 - The three methods by which you can repeat a part of a program are,
 - By using a while loop
 - By using a do-while loop
 - By using a for loop

While LOOP:-

- ✚ The simplest of all the looping structures in C is the while loop.
- The general form (or) the syntax of while loop is:

```
initialize loop counter;  
while(condition)  
{  
    Statement(s);  
    increment (or) decrement loop counter;  
}
```



- The above figure shows the operation of the while loop.
 - A looping process generally has three steps. They are,
 - ❖ Initialization of the counter
 - ❖ Test for a specified condition for execution of the statements in the loop
 - ❖ Incrementing (or) Decrementing the counter.
- In looping, a sequence of statements is executed until some conditions are satisfied.
 - A loop consists of two segments:
 - ❑ One is referred as the body of the loop and the other is
 - ❑ Control statement.
- The control statement tests certain conditions and then directs the repeated execution of the statement(s) contained in the body of the loop structure.
- When the condition specified inside the parenthesis of while loop is satisfied, the control is transferred to the statement(s) inside the loop and executes the body of the loop and increments (or) decrements the loop counter.
- The loop is continues when the condition is violated.
- The while loop tests the condition before each iteration.
- If the condition initially fails the loop is skipped entirely even the first iteration itself.

do while LOOP:-

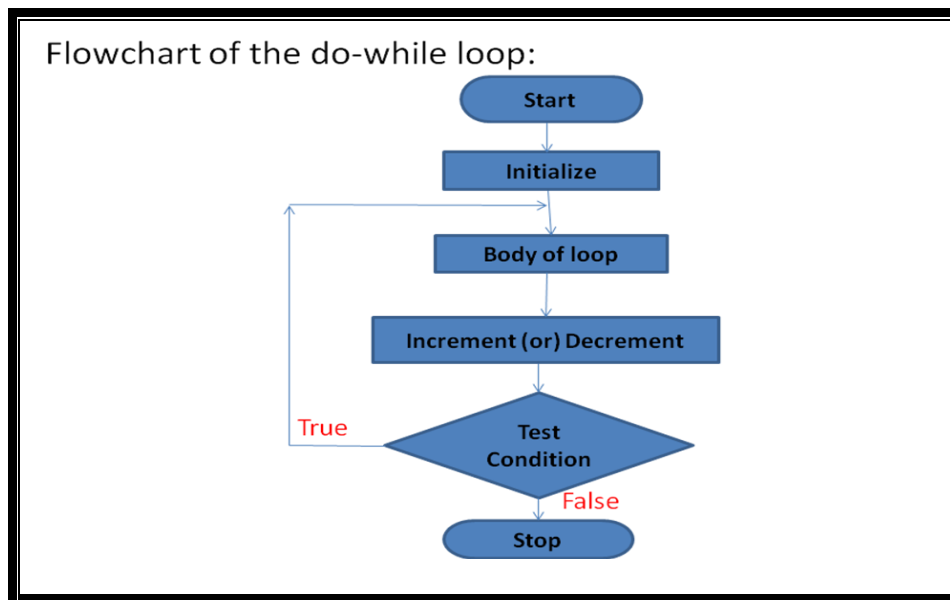
- The do-while loop sometimes referred as “**do loop**” differs from its counterpart the while loop in checking the condition.
 - The condition of the loop is not tested until the body of the loop has been execute once.
 - If the condition is false, after the first loop iteration the loop terminates.
 - However if the condition is true the loop continues.
- The general form (or) the syntax of the do-while loop is:

```

initialize loop counter;
do
{
    Statement(s);
    increment (or) decrement loop counter;
}while(condition);

```

- In do-while the statements would be executed at least once even if the condition fails for the first time itself.



- The control enters the loop without testing any condition and the body of the loop is executed once.
- After executing once, the while statement it checks the condition, since the condition is true control is transferred again to the starting of the do loop and the body of the loop is again executed.
- This process continues until the condition is false.

for Loop:-

- The for loop allows us to specify three things about the loop in a single line. They are,
 - ✚ Initializing the value for the loop
 - ✚ Condition in the loop counter to determine whether the loop should continue (or) not.
 - ✚ Incrementing (or) decrementing the value of loop counter each time the program segment has been executed.
- The general form (or) the syntax of the for loop is:

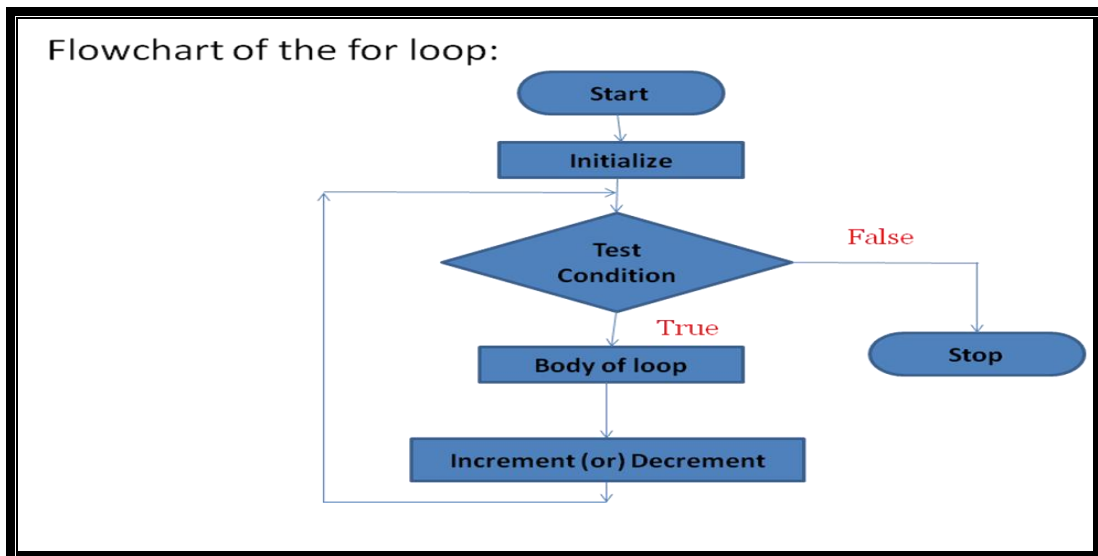
```

for(e1; e2; e3)
{
    statement(s);
}

```

Where

- **e1** is an expression which is used to initialize one (or) more variables used in the for loop. (**Initialization**)
- **e2** is an expression which is used to test the condition in the for loop. (**Condition**)
- **e3** is another expression used for incrementing the index which may/may not be initialized in the expression. (**Increment (or) Decrement**)
- When the for loop starts its execution the expression(s) present in **e1** will be initialized first and then checks for the condition in expression **e2**.
- If the condition satisfies, the statement(s) in the body of the loop is executed once and then increments (or) decrements the index as in expression **e3**.
- The looping action continues as long as the condition in **e2** is true.
- Note that the control goes to expression **e1** only when it enters loop for the first time only, after that depending upon the condition the expressions **e2** and **e3** are executed alternatively.



The break Statement

- The break statement is used to terminate (or) exit from switch statement.
- The general form (or) the syntax of the break statement is
break;
- The break statement does not have any embedded expression (or) arguments.
- The break statement can be also used within a for, while (or) do-while loops.
- The break statement is usually used at the end of each case and before the start of the next case statement.
- The keyword break breaks the control only from the loop in which it is placed.

Ex. Program:

```

/* break .c */
#include<stdio.h>
#include<conio.h>
void main( )
{
    int i,n;
    clrscr( );
    for(i=0; i<100; i++)
    {
        printf("%d\t", i);
        if(i==10)
        {
            break;
        }
    }
}
  
```

Output:

0 1 2 3 4 5 6 7 8 9 10

- ❑ The above program prints the numbers 0 through 10 on the monitor. Then the loop terminates because break causes immediate exit from the loop. Overriding the conditional test $i \leq 100$.

The continue statement:-

- The continue statement is used to transfer the control to the beginning of the loop, there by terminating the current iteration of the loop and starting again from the next iteration of the same loop.
- The continue statement can be used within a while loop (or) do-while (or) for loop.

- The general form (or) syntax of the continue statement is:

continue;

- The continue statement does not have any expressions (or) arguments.
- Unlike break, the loop does not terminate when a continue statement is encountered, but it terminates the current iteration of the loop by skipping the remaining part of the loop and resumes the control to the start of the loop for the next iteration.
- The continue statement is usually used within an iteration statement.
- It causes the control to pass to the loop-continuation portion in which the continue statement is enclosed.
- If a continue statement is not enclosed within a loop, a syntax error is indicated by the compiler.
- The continue statement is only apply for loop not for switch statement.

Ex.Program:

```
/* continue .c */
#include<stdio.h>
#include<conio.h>
void main( )
{
    int i;
    clrscr( );

    for(i=1;i<=5;i++)
    {
        if(i==3)
            continue;
        printf("%4d", i);
    }
}
```

Output:

1 2 4 5

- In above program

- ❑ When the value of i equals to that of 3, the continue statement takes the control to the for loop without executing to the printf() function.

The goto statement:-

- The goto statement is used to transfer the control in a loop (or) a function from one point to any other portion in that program.
- If misused the goto statement can make a program impossible to understand.
- The general form (or) the syntax of goto statement is

goto lable;

statement(s);

```
. . .
. . .
. . .
. . .
```

label:

statement(s);

- The goto statement requires a label in order to identify the place where the control is to be transferred.
- A label may be any valid variable name.

Ex. Program:

```
/* goto .c */
#include<stdio.h>
#include<conio.h>
void main( )
{
    int runs;
    clrscr( );
    printf("Enter the runs scored by India:");
    if (runs <= 300)
    {
        goto pitch;
    }
    else
    {
        printf("India will win the match");
        exit( );
    }
    pitch:
    printf(" India may loose the match");
}
```

Output:

RUN 1: Enter the runs scored by India: 350
India will win the match
RUN 2: Enter the runs scored by India: 250
India may loose the match

- The label must be followed by a colon(:).
- The label can be placed any where in the program either before after the goto statement.
- The goto statement breaks the normal sequential execution of a program.
- It merely instructs the computer to resume execution at a specified statement, which is specified as a label.
- The naming rules for labels are the same as those for variables except that label is not declared.

In above Program:

➤ In RUN 1,

- As soon as the user enters 350, it is assigned to the variable runs.
- The if statement checks the condition, whether the value is lesser than (or) equal to 300.
- Since , the condition is false in this case, the statement in the if block is skipped and the statements in the else block is executed, and printf() function displays the message

India will win the match

- and the program ends, because of the use of exit() function.

➤ In RUN 2,

- As soon as the user enters 350, it is assigned to the variable runs.
- The if statement checks the condition, whether the value is equal to 300.
- Since, the condition is true in this case, this statement in the if block executed, and the control from the if block is transferred to the lable(i.e; pitch).
- The printf() function below the label will be executed and the printf() will be displays the message

India may loose the match

And the program ends.

LECTURE 10:

CONCEPTS OF ARRAYS & TYPES OF ARRAYS***Arrays***

- ❖ C language provides the capability that enables the user to design a set of similar data types called “**arrays**”.
 - They are defined in the same manner as ordinary variables except that each array name must be accomplished by size specification within square braces capable of storing more than one value at time.
 - The array is defined as a set of homogeneous data items of the same type that share a common name.
 - The individual values in the array are called “**elements**”.
 - Each array element is referred by specifying the array name, followed by a number within square braces referred as an “**index (or) subscript**”.
 - The dimension of an array is determined by the number of subscripts needed to identify each element.
 - A subscript must be always enclosed in square braces[], placed after the array name.
 - Each subscript must be expressed as a non-negative integer.

Declaring Arrays:-

- Like an ordinary variable, an array variable should also be declared.
 - The general form (or) syntax of declaring an array is:

datatype arrayName[Subscript];

Where

- arrayName** is a valid C variable of type **datatype**
- subscript** is an integer constant indicating the maximum number of elements that can be stored in the array.

Ex:

int days[31];

is read as “days,an array of 31 int values”.

- In the above declaration:
 - int**-data type
 - days**- name of the array
 - 31**-size of the array

Properties of arrays:-

- + The type of an array is the data type of its elements.
 - + The location of an array is the location of its first element.
 - + The length of an array is the number of data elements in the array.
 - + The size of the array is the length of the array , but the size of the array is usually referred as the product of subscripts used.
 - + An array name should be a valid C identifier.
 - + The name of the array should be unique, similar to other variables.
 - + The value of the elements stored in the array should be of the same type(i.e; if an array is declared as int, it can store elements that are only int).
- **Important points to be noted while using subscripts:**
- A subscript value should not be negative.
 - A subscript must always be an integer (or) an expression that gives integer.
 - A subscript must be specified within the square brackets preceding the array name.
 - If there are more than one subscript as in the case of multi dimensional arrays each subscript must be specified within square brackets.

Types Arrays:-

- **The arrays are classified into three types, they are:**
- Single Dimensional Arrays(One Dimensional Arrays)
 - Double Dimensional Arrays(Two Dimensional Arrays)
 - Multi Dimensional Arrays
- This classification is mainly depends on the number of subscripts are used in the declaration of an array.

Single Dimensional Arrays:-

- Arrays whose elements are specified by a single subscript are called as “**One-dimensional (or) Single dimension (or) single-subscripted (or) linear arrays**”.
- The arrays that have seen so far are all **single dimensional arrays**.
- A list of items can be given a variable name using only one subscript and such a variable is called as a single dimensional array also referred as “**list**”.
- The main purpose of an array is to reserve space for the elements in memory.

- The general form (or) syntax of an array declaration is:

storageType dataType arrayName[Size];

Where

- storage Type** while declaring an array is optional.
- The **dataType** specified the type of elements that will be stored in the array.
- The **arrayName** is the name used to identify the array.
- The rule for naming arrays are same as identifiers.
- The size is a positive integer constant, indicating the maximum number of elements that can be stored in the array.
- Each subscript must be expressed as a non-negative integer.
- In an n-element array, the array elements are stored in x[0],x[1],...x[n-2],x[n-1] order.

Ex:

```
int value[50];
char book[104];
```

- In the first example, **int** specifies the type of the variable and **value** specifies the name of the array.
- The number **50** within square braces indicates the maximum number of elements to be stored in the array.
- In second one the **book** is character array with a size of **104** elements.

Double Dimensional Arrays:-

- Array whose elements are specified by two subscripts are referred as **two-dimensional arrays** (or) **double dimensional arrays**.
- The double dimensional array is defined in the same manner as single dimensional array except that it requires two pairs of square brackets for two subscripts.
- The double dimensional array is also referred as **matrix**.
- Each subscript must be expressed as non negative integer.

- The general form (or) syntax of a double (or) two dimensional array is:

storageType dataType arrayName[rowsize][colsize];

Where

- storage Type** while declaring an array is optional.
- The **dataType** specified the type of elements that will be stored in the array.

- ❑ The **arrayName** is the name used to identify the array.
- ❑ The rule for naming arrays are same as identifiers.
- ❑ The rowsize and colsize are positive value integer constants.
- ❑ These two subscripts indicate, the number of array elements associated with each subscript.
- ❑ Two square braces are used to indicate the values of two subscripts.

Ex:

int mark[2][2];

- The above array declaration defines the table as an integer array having 2 rows and 2 columns.
- An array elements start with an index zero and so the individual elements of the array will be

$$\text{int mark} \left(\begin{array}{cc} [0][0] & [0][1] \\ [1][0] & [1][1] \end{array} \right)$$

- The first element stored in the memory location [0][0], followed by the second element in [0][1] and so on, and the last element will be stored in the memory location [1][1].
- An **m X n**, double dimensional array can have a table of values having **m** rows and **n** columns.
- It is not compulsory that the row-size and col-size should be equal.
- Care must be taken to initialize the values of the elements.

LECTURE 11:

TYPES OF FUNCTIONS (LIBRARY FUNCTIONS & USER DEFINED FUNCTIONS)***Types Of Functions***

- ❖ A function is a self contained program segment (block of segments) that performs some specific well-defined task when called.
 - Functions break large computing tasks in to smaller ones.
 - C allows programmers to define their own functions for carrying out various individual tasks.
 - C has been designed to make functions efficient and easy to use.
 - C programs generally consist of many small functions rather than few big ones.
 - Every C program consists of one or more functions, but there should be at least one function which must be main().
 - Each time when a new program is written, main() function must be defined.
 - The main () function calls another function to share the work.
 - C functions may be classified into two categories namely
 - Library functions
 - User-defined functions

Library Function

- ❖ C's standard library provides a rich collection of functions for performing input, output, string manipulations, common mathematical calculations, and many other functions for performing useful operations.
 - There are many library:
 - ✚ Functions that carries standard input and output operations
 - Ex:** scanf(), printf(), gets(), puts() etc..,
 - ✚ Functions that perform operations on characters
 - Ex:** tolower(), toupper(), etc..,
 - ✚ Functions that perform operations on strings
 - Ex:** strcpy(), strcat(), strcmp(), strlen(), etc..,
 - ✚ Functions that perform mathematical calculations
 - Ex:** sin(), cos(), sqrt(), pow(), etc..,
 - A library function is accessed simply by writing the function name followed by a list of arguments, which represents the information being passed to the function.
 - The arguments must be enclosed in parenthesis and separated by commas.

Used Defined Function

- ❖ These functions are defined by the user according to his requirements.
- ❖ The user can implement his ideas in developing user-defined functions.
 - The main() function that we have seen so far in the programs is an example of an user-defined function.
 - The main difference between these two categories is that, an user-defined function has to be developed by the user at the time of writing a program, where as library functions are previously defined functions.
 - A function can be called by simply using the function name as a statement.
 - A function will carry out its intended action whenever it is called from some other portion of the program. The same function can be accessed (called) from different places within a program.
 - Once the function has carried out its intended action, control will be returned to the next statement from which the function is accessed.
- **The above example program helps you to understand things better.**

- As we know that the execution of a C program starts from the main() function, the first printf() function executes and displays the message:

Inside main function

- And the control is transferred to the next statement, the function call statement (i.e., func()).
- When the compiler encounters a function call, control is transferred to that function.

- The function is executed and displays the message:

Inside func function

- Once the function func() has completed its execution, control is transferred or returned to the next statement from which the function is called (i.e., to the second printf() function inside the main() function).

- The printf() function displays the message:

Again inside main function

and the program ends.

Ex. Program:

```

/* function. c */
#include<stdio . h>
void func( ) ;
void main( )      /* function prototype */
{
    printf("\n Inside main function");
    func( ) ; /* function call statement */
    printf("\n Again inside main function");
}
void func( ) /* function definition */
{
    printf{"\n Inside func function"};
}

```

Output:

```

Inside main function
Inside func function
Again inside main function

```

Note:

✚ The semicolon is present in the function, when the function is called within the main() function.

- This is compulsory when the function is called from another function.
- Generally, a function will process information that is passed to it from the calling part of the program.
- Information is passed to the function via special identifiers called **arguments or parameters** and returned via the return statement.

Advantages of Functions:-

- There are many advantages in using functions for writing programs.
- For example, many programs may require a particular group of statements to be executed repeatedly from different places within the program. These repeated statements can be placed within a single function, which can be executed whenever it is needed.
- Thus, the use of a function avoids the need for same statements to be repeated in a program and thus saves memory.
- The use of functions also enables a programmer to build user-defined library functions of frequently used routines containing system dependent features. Each routine can be programmed as a separate function and stored within a special library file.
- These user-defined library functions can be added as a later part of the C programming library.

LECTURE 3:

CONCEPT OF FUNCTIONS (DEFINING A FUNCTIONS & FUNCTION PROTOTYPE)**Defining a Function:-**

- A function definition consists of two parts. They are,

- ✚ Argument declaration
- ✚ Body of the function

- The first part of the function specification consists of type specification of the value returned by the function followed by the function name, a set of arguments (may or may not be present) separated by commas and enclosed in parenthesis.
- If the function definition does not include any arguments, an empty pair of parenthesis must follow the function name.
- The general form or the syntax of defining a function is,

```

returnType functionName (argument list)
{
    declaration (s) ;
    statement (s);
    return(expression);
}

```

} Body of the function

Where

- The returnType specifies the data type of the value to be returned by the function. C assumes that every function will return a value.
- The returnType is assumed to be of type int by default if it is not specified.
- The functionName is used to identify the function.
- The rules for naming a function are same as identifiers.
- The argument list specified inside the parenthesis after the function name is optional.
- It contains valid variable names with their data types preceding them.
- Semicolons are not allowed after the closing parenthesis, while defining a function.
- Most functions have list of parameters and a return value that provides means for communication between functions.
- The arguments in the function reference, which defines the data items that are actually transferred, are referred as **actual arguments or actual parameters**.
- The arguments that represent the names of data items that are transferred to the function from the calling portion of the program are referred as **formal arguments (or) formal parameters**.
- All variables declared in function definitions are local variables.
- Their scope is visible known only in the function in which they are defined. Functions arguments are also local variables.

- The second part of the function definition contains a statement or a group of statements that defines the action to be done by the function.
- These statements starting from the opening brace of the function to the last statement before the closing brace of the function is called as the **body of the function**.
- The variables used in the function must be declared in the start of the function following the opening brace of the function (or) before all expression statements.

For example:

```
int add(int x, int y)  /* argument declaration */
{
    int z;
    z =x + y;
    return (z);
}
```

} **Body of the function**

- The function named **add()** takes up two integers namely x and y.
- The function returns an integer value, since int is specified before the function name **add**.
- The data type of the arguments passed inside a function may be specified in the declaration (or) before the opening brace of the function body.

For example:

```
int add(x, y)  /* argument declaration */
int x, y;
{
    int z;
    z=x+y;
    return ( z );
}
```

} **Body of the function**

is also valid.

- On executing a return statement, control of the program returns to the calling function.
- The expression within parenthesis of the return statement is the value, which is returned to the called function.
- If the function reaches the final closing braces without any return statement, control will be transferred to the calling function without passing any value.
- The return type specified in this case is void.
-

The return Statement

- The **return** statement is used to return the control from the calling function to the next statement of the called portion of the program.
- The **return** statement also causes the program logically to return to the point from where the function is accessed (called).
- The **return** statement returns one value per call.

- The **return** statement can be any one of the types as shown below:

```
return;
return();
return(constant);
return(variable);
return(expression);
return(conditional expression);
return(function);
```

- The first and second **return** statements, does not return any value, and are just equal to the closing brace of the function.
- If the function reaches the end without using a return statement, the control is simply transferred back to the calling portion of the program without returning any value.
- The presence of an empty return statement (without any expression or constant or variable) is recommended in such situations.
- These **return** statements returns a value 1 to the calling function.
- The third **return** statement returns a constant to the calling function.

For example:

```
if(x <= 1)
return(1);
```

- The return statement returns a constant 1 when the condition specified inside the if statement evaluates to true.
- The fourth return statement returns a variable to the calling function.

For example:

```
if(x <= 15)
return(x);
```

- The **return** statement returns a variable (which may return any value) to the calling function depending upon the value of the variable x.
- The fifth **return** statement returns a value depending upon the result of the expression specified inside the parenthesis.

For example:

```
return(a +.b * c) ;
```

- returns a value depending upon the values of a, b.and c.
- The sixth **return** statement returns a value depending upon the result of the conditional expression specified inside the parenthesis.

For example:

```
return(a > b ? a : b);
```

- The above return statement returns a value depending upon the value of the variables a and b.
- The last return statement calls the function specified inside the parenthesis and collects the result obtained from that function, and returns it to the calling function.

For example:

```
return(power(3,2));
```

- returns a value, obtained after evaluating the power() function.

Important points to be noted while using return statement:-

- The limitation of a return statement is that it can return only one value from the called function to the calling function.
- The return statement can be present anywhere in the function, not necessarily at the end of the function.
- Number of return statements used in a function are not restricted, since the return statement which executes first will return the value from the called function to the calling function and the other return statements are left unexecuted.
- If the called function does not return any value, then the keyword void must be used as the return type specifier.
- Parenthesis used around the expression in a return statement is optional.

Function Prototypes:-

- A C function returns an integer value by default.
 - Whenever a call is made to a function the compiler assumes that this function would return a value of type int.
 - If you decide that a function should return a value other than int, then it is necessary to mention the first line of the function in the program, before it is used, which is called as the **function prototype** also referred as the **function declaration**.
 - The function prototype not only identifies the return type of the function, but also the name of the function, the number of parameters in the function, the data types of the parameters passed and the order in which they are expected.
 - Function prototypes are usually written at the beginning of the program explicitly before all user-defined functions including the main () function.
- The general form or the syntax of declaring a function prototype is,

```
returnType functionName(dt1 arg1,dt2 arg2,....,dtn argn);
```

Where

- returnType represents the data type of the value that is returned by the function, and dt1,dt2, . . . , dtn represents the data types of the arguments arg1, arg2, . . . , argn.
- The data types of the arguments should be specified compulsorily in the function prototype, but specifying the arguments of the data types is optional.
- The function prototype resembles the first line of the function definition (but the first line of the function definition does not end with a semicolon).
- The names of the arguments within the function prototype need not be declared elsewhere in the program.
- The data types of the actual arguments must confirm to the data types of the arguments within the prototype.

For example:

```
int sum(int x);
```

Where

- sum** is the name of the function, **int** before the function name `fact()` indicates that the function returns a value of type **int**.
- The variable **x** inside the parenthesis is the parameter passed to the called function.
- The data type **int** before the parameter **x** indicates that it is of type integer.

* Types of Functions *

➤ A function depending upon the arguments present or not and whether a value is returned or not, may be classified as

1. Function with no arguments and no return values.
2. Function with return values and no arguments.
3. Function with arguments and no return values.
4. Function with arguments and return values.

1. Function with no arguments and no return values:

- When a function has no arguments, it does not receive any data from the calling function, similarly when a function has no return values, the calling function does not receive any data from the called function.
- Hence, there is no data transfer between the calling function and the called function.
- If such functions are used to perform any operation, they read the input and display the output in the same block.

Ex. Program:

```
#include<stdio.h>
#include<conio.h>
void name( ); /* function prototype */
void main( )
{
    name( );
}
void name( )
{
    char empname[25];
    printf("Enter the employee name :");
    scanf("%s", empname);
    printf ( "The employee name is %s", empname);
}
```

output:

```
Enter the employee name : Rama
The employee name is Rama
```

In Ex.Program:

- main ()** is the calling function which calls the function **name()**.
- The function **name()** contains no arguments and hence, there are no argument declarations.
- Note that the called function (i.e., **name**) receives its data (i.e., name of the employee) directly from the input terminal (i.e., keyboard) and displays the contents of **empname** to the output terminal (i.e., screen) in the called function itself.
- No **return** statement is employed since there is nothing to be returned.

- ❑ The closing brace of the function indicates the end of execution of the function, thus returning the control, back to the calling function.
- ❑ The keyword **void** is used before the function **name()** to indicate that there are no return values.

2. Function with retrun values and no arguments:

- When a function has no arguments, it does not receive any data from the calling function, but it can do some process and then return the result to the called function.
- Hence, there is data transfer between the calling function and the called function. .

In Ex.Program:

- ❑ **main ()** is the calling function which calls the function **sum ()** .
- ❑ The function **sum ()** contains no arguments and hence, there are no argument declarations.
- ❑ Note that the called function (i.e., **sum ()**) receives its data (i.e., the limit for calculating the even numbers) directly from the input terminal (ie., keyboard) in the called function (i.e., **sum ()**).
- ❑ The **return** statement is employed in this function to return the sum of first n even numbers calculated within the limit and the result is displayed from the **main ()** function to the standard output device (i.e., screen).
- ❑ Note that **int** is used before the function name **sum ()** instead of **void** since it returns a value of type **int** to the called function.

Ex. Program:

```
/* Program to find the sum of even numbers
within the limit */
#include<stdio.h>
#include<conio.h>
int sum( );          /* function prototype */
void main( )
{
    printf ( " is %d" , sum ( ) ) ;
}
int sum( )
{
    int i, n, result=0;
    printf("Enter the limit : ");
    scanf("%d", &n) ;
    printf("Sum of Even numbers within %d" ,n);
    for(i=2;i<=n;i+=2)
    {
        result += i;
    } return(result);
}
```

Output:

```
Enter the limit : 10
Sum of Even numbers within 10 is 30
```

3. Function with arguments and no retrun values:

- When a function has arguments, it receives data from the calling function.
- The **main()** function will not have any control over the way in which the functions receives its-input data.
- We can also make the calling function to read data from the input terminal and pass it to the called function.
- This approach seems better because the calling function can check the validity of data, before it is passed over to the called function.

Ex. Program:

```

/* Program to calculate simple interest*/

#include<stdio.h>
#include<conio.h>
void simple(float , float, int);/* function prototype*/
float principal, rate;
int months;
void main( )
{
printf ("Enter principal, rate of interest & no. of months\n");
scanf("%f %f %d" , &principal, &rate, &months) ;
simple(principal, rate, months);
}
void simple (float principal, float rate, int months)
{
float simple interest =0.0;
simple_interest = ( principal * rate * months ) / 100;
printf ("The simple interest is %.2f", simple_interest);
}

```

Output:

```

Enter principal, rate of interest & no. of months
5000 1.5 2
The simple interest is 150.00

```

In above Ex.Program:

- main()** is the calling function which calls the function **simple()**.
- The function receives three arguments (i.e., **principal, rate** and **months**).
- The first two arguments are of **floating** type and the third argument is of **integer** type.
- Note that the called function (i.e.,**simple()**) receives its data from the calling function (i.e.,**main()**).
- No **return** statement is employed since there is nothing to be returned.
- The closing brace of the function signals the end of the function thus returning the control back to the calling function.
- The keyword **void** is used before the function name **simple()** to indicate that there are no return values.

4. Function with arguments and retrun values:-

- When a function has arguments it receives data from the calling function and does some process and then returns the result to the called function.
- In this way the main 0 function will have control over the function.
- This approach seems better because the calling function can check the validity of data before it is passed to the called function and to check the validity of the result before it is sent to the standard output device (i.e.,screen).

- Note that when a function is called, a copy of the values of actual arguments is passed to the called function.

Ex. Program:

```

/* Program to find the sum of N natural numbers*/

#include<stdio.h>
#include<conio.h>
int sum (int x) ;                               /* function prototype */
void main( )
{
    int n;
    printf("Enter the limit : ");
    scanf("%d", &n);
    printf("Sum of first %d natural numbers is %d", n, sum(n));
}
int sum(int x)
{
    int i,result = 0;
    for(i = 1 ; i <= x ; i++)
        result += i;
    return(result);
}

```

Output:

```

Enter the limit : 5
Sum of first 5 natural numbers is 15

```

In Ex.Program:

- main()** is the calling function which calls the function **sum()**.
- The function **sum()** receives a single argument.
- Note that the called function (i.e., **sum()**) receives its data from the calling function (i.e., **main()**).
- The **return** statement is employed in this function to return the sum of the n natural numbers entered from the standard input device and the result is displayed from the **main()** function to the standard output device.
- Note that **int** is used before the function name **sum()** instead of void since it returns the value of type **int** to the called function.

Important points to be noted while calling a function:-

- ✚ Parenthesis are compulsory after the function name.
- ✚ The function name in the function call and the function definition must be same.
- ✚ The type, number, and sequence of actual and formal arguments must be same.
- ✚ A semicolon must be used at the end of the statement when a function is called.

Nested Functions

- Previously we have seen programs using functions called only from the `main()` function.
- But there are situations, where functions other than `main()` can call any other function(s) used in the program. This process is referred as **nested functions**.

In Ex.Program:

- ❑ Uses two functions **func1()** and **func2()** other than the **main()** function.
- ❑ The **main()** function calls the function **func1()** and the function **func1()** calls the function **func2()**.
- ❑ The output of the program explains the order of execution of the functions used in the program.

Ex. Program:

```
/* Program to demonstrate the use of nested functions*/
```

```
/* nest_fun.c */
```

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

```
#include<conio.h>
```

```
void func1( );
```

```
void func2( );
```

```
void main( )
```

```
{
printf("\n Inside main function");
```

```
func1( );
```

```
printf("\n Again inside main function");
```

```
}
```

```
void func1( )
```

```
{
```

```
printf("\n Inside Function 1");
```

```
func2( );
```

```
printf("\n Again inside Function 1");
```

```
}
```

```
void func2( )
```

```
{
```

```
printf("\n Inside Function 2"); , .
```

```
}
```

Output:

```
Inside main function
```

```
Inside Function1
```

```
Inside Function2
```

```
Again inside Function 1
```

```
Again inside main function
```

Recursion

- In C it is possible to call a function itself.
- **Recursion** is a process by which a function calls itself repeatedly until some-specified condition has been satisfied.
- A function is called recursive if a statement within the body of a function calls the same function.
- Recursion is a process of defining something in terms of itself.

➤ **The following example helps you to understand things better:**

Ex. Program:

```

/* Program to find factorial of a number using
recursive function*/

/* fact_rec.c */
#include<stdio.h>
#include<conio.h>
long fact (long);      /* function prototype */
void main( )
{
    long num,  fac;
    printf("Enter a number : ") ;
    scanf("%ld", &sum) ;
    fac = fact (num) ;
    printf("The factorial of %ld is %ld", num, fac);
}
long fact(long num)
{
if(num <= 1)
    return(1);
else
    return(num * fact(num- 1));
}

```

Output:**RUN 1:**

```

Enter a number : 0
The factorial of 0 is 1

```

RUN2:

```

Enter a number : 3
The factorial of 3 is 6

```

In **RUN1**,

- When the number entered is **0**, let us see what action the function **fact()** does.
- The value of the parameter num (i.e., **1**) is copied into **num**.
- Since **num** turns out to be **0** the condition (**num<= 1**) is satisfied and hence **1** is returned through the return statement (i.e., the factorial value of **0** is **1**), and the printf() function displays the message The factorial of 0 is 1.

In **RUN2**,

- When the number entered is **3**, the condition (**num<=1**) fails, and hence the control is transferred to the else block.

num * fact (num-1);

- Since the current value of **num** is **3**, the above statement would be

3 * fact (2) ;

will be evaluated.

- The expression on the right-hand side includes a call to the function **fact()** with **num** equal to **2**, which returns

2 * fact(1) ;

- Once again the function **fact()** is called with **num** equal to **1**.

- This time, the function returns **1**.

- The sequence of operations can be summarized as

$$\begin{aligned} \text{fac} &= 3 * \text{fact} (2) \\ &= 3 * 2 * \text{fact} (1) \\ &= 3*2*1 \\ &= 6 \end{aligned}$$

➤ Important points to be noted while using functions:-

- ✚ C program may contain one (or) more functions.
- ✚ A function can be called any number of times.
- ✚ A function can be called from any other function.
- ✚ The called function returns only one value per call.
- ✚ A function declaration must end with a semicolon.
- ✚ A function definition should not end with a semicolon.
- ✚ A function cannot be defined within another function definition.
- ✚ A function can call itself this type of process is called recursion.
- ✚ A function cannot be declared and defined more than once in a program.
- ✚ A function definition may appear in any order (i.e., before or after the main() function).
- ✚ The function name in the function call and the function definition should be the same.
- ✚ The order in which functions are defined' in a program and the order in which they are called need not be the same.
- ✚ If a function takes no arguments a parenthesis after the function name is a must.
- ✚ The type, order, and number of actual arguments must be the same as that of the formal arguments.

Advantages of recursive functions:-

- Used to create clearer and simpler versions of several algorithms.
- Memory occupied when the recursive function is called is very less compared to an ordinary function.
- Recursion is a very useful way of creating and accessing certain dynamic data structures such as linked lists, stacks, queues, etc.,

LECTURE 13:

CONCEPT OF STRING LIBRARY FUNCTIONS***String Library Functions***

- ❖ With every C compiler a large set of useful library functions are provided.
- The following **table** shows the commonly used string functions along with the purpose of their usage.

FUNCTION	MEANING
strcpy()	Copies a string into another string
strcat()	Appends one string at the end of another string
strchr()	Finds first occurrence of a given character in a string
strcmp()	Compares two strings
strncmpi()	Compares two strings without regard to case("i" denotes that this function ignores case)
strcpy()	Copies a string into another string
strcspn()	Finds the initial segment of string si consists entirely of characters not from string s2.
strdup()	Duplicates a string
stricmp()	Compares two strings without regard to case (identical to strcmpi)
strlen()	Finds length of a string
strlwr()	Converts a string to lowercase
strncat()	Appends first h characters of a string at the end of another string
strncmp()	Compares first n characters of two strings
strncpy()	Copies first n characters of one string into another string
strnicmp()	Compares two strings without regard to case (identical to strcmpi)
strnset()	Sets first n characters of a string to a given character
strpbrk()	Scans a string, si, for the first occurrence of any character appearing in s2
strrev()	Finds last occurrence of a given character in a string Reverse a string.
strset()	Sets, all characters of string to a given character

FUNCTION	MEANING
strspn()	Finds the initial segment of string <i>s1</i> that consists entirely of characters from string <i>s2</i> .
strstr()	Finds first occurrence of a given string in another string
strtok()	Scans <i>s1</i> for the first token not contained in <i>s2</i>
strupr()	Converts a string to uppercase
strxfrm()	Transforms the string <i>*s2</i> into the string <i>*s1</i> for no more than <i>n</i> characters.

String Library Functions

✚ From the above list we shall discuss the functions **strlen()**, **strcpy()**, **strcat()**, **strcmp()**, **strrev()**, **strlwr()**, **strupr()** since these are the most commonly used string library functions.

✚ With every C compiler a large set of useful library functions are provided.

➤ The following are most commonly used string library functions, they are:

- strlen()** (String Length)
- strcpy()** (String Copy)
- strcat()** (String Concatenate)
- strcmp()** (String Compare)
- strrev()** (String Reverse)
- strlwr()** (String Lower)
- strupr()** (String Upper)

The strlen() Function:-

- The **strlen()** (string length) function returns the count of number of characters stored in a string.

➤ The general form (or) syntax of **strlen()** function is

X = strlen(str);

Where

- **X** is an integer variable which receives the length of the string and **str** is a valid string variable (or) a constant.

The strcpy() Function:-

- A string cannot be copied directly using an assignment statement (with the exception of pointer).
- However, it is possible to copy a string element by using assignment statements.
- The **strcpy()** (string copy) function is used to copy the contents of one string to another.
 - The general form (or) syntax of **strcpy()** function is:

strcpy(string1, string2);

Where

- **String1** and **string2** are valid string variables (or) string constants.
- The **strcpy()** function takes two arguments for copying.
- The string in second argument (i.e.; **string2**) is copied to the first argument (i.e.; **string1**).

The strcat() Function:-

- The **strcat()** (string concatenate) function concatenates the source string at the end of the target string.
- The **strcat()** function takes two arguments for concatenating.
- The string in the second argument (i.e., **string2**) is concatenated with the string present in the first argument (i.e.; **string1**).
 - The general form (or) the syntax of **strcat()** function is

strcat(string1, string2);

Where

- **string1** and **string2** are valid string variables (or) string constants.
- The content of **string2** is concatenated with the content of **string1**.

The strcmp() Function:-

- The **strcmp()** (string compare) function compares two strings to find whether the strings are equal (or) not.
 - The general form (or) syntax of **strcmp()** function is:

strcmp(string1, string2);

Where

- **string1** and **string2** are valid string variables (or) string constants.
- The **strcmp()** function takes two arguments for comparing and returns an integer.

- The string in the first argument(i.e., **string1**) is compared with the string in the second argument(i.e; **string2**) character until it encounters a mismatch between the two strings (or) end of one of the two strings.

The **strrev()** Function:-

- The **strrev()** (string reverse) function is used to reverse a string.
- This function takes only one argument.
 - The general form (or) syntax of **strrev()** function is

X= strrev(string);

Where

- **X** is the string variable to hold the reversed string and string is the valid string variable (or) string constant which stores the string to be reversed.

The **strlwr()** Function:-

- The **strlwr()** (string lower) function is used to convert a string to lowercase.
- This function takes only one argument
 - The general form (or) syntax of the **strlwr()** function is:

X= strlwr(string);

Where

- **X** is the string variable to hold the lowercase converted string and string is the valid string variable (or) string constant which contains the string(**uppercase**).

The **strupr()** Function:-

- The **strupr()** (string upper) function is used to convert a string to uppercase.
- This function takes only one argument.
 - The general form (or) syntax of the **strupr()** function is:

X= strupr(string);

Where

- **X** is the string variable to hold the uppercase converted string and string is the valid string variable (or) string constant which contains the string(**lowercase**).

LECTURE 14:

CONCEPT OF POINTERS*Pointer*

- We know that variables are stored in memory. Each memory location has a numeric address.
- Variable names in C have the capability to enable the programmer to refer to the memory location by its name, but the compiler must translate these names into their respective memory addresses.
- This process is automatic and the programmer need not be worried about it.
- The address or name of a memory location points to whatever it contains within that memory location.
- Passing addresses does not violate the rules of programming.
- The addresses are passed only to the functions that need to access those memory locations.
- At any moment, in a program execution, each existing variable has a unique address associated with it.
- The memory location formerly occupied by such a local variable is freed when the program execution ends and the memory location is allocated to another variable later in some other program.
 - ❖ A **pointer** is a variable, which represents the location (not the value) of a data item, such as a variable or an array element.

Pointer Operator:-

- Like all variables, a pointer variable should also be declared.
- The general form or the syntax of a pointer variable is

dataType *variableName ;

where

- ✚ dataType is the type of the variableName, which may of any valid type.
- ✚ * is called as **the indirection operator** also referred to as the **dereferencing operator**, which states that the variable is not an variableName is the name of the pointer variable.

- The rules for naming pointer variables are same as Identifiers.
- When an * is placed before the variableName, the value returned by it, is not the value of the pointer variable, but the address of the value stored in the memory location.
- The content of any pointer variable can be accessed with the help of indirection operator.
- If **ptr** is a pointer variable, then ***ptr** is used to access, the content of the pointer variable **ptr**.
- The indirection operator (*) can only act upon operands, which are pointer variables.
- All the variables declared in a program occupy a specific address in memory.

- It is possible in C, to obtain the address of the variable using an operator referred as the **address operator**.
 - This operator may precede a variable name or an array element, but should not precede a constant, an expression, or an un-subscripted name of an array.
 - Let **ptr** be a variable that represents a data item. The compiler automatically assigns memory cells for the data item.
 - The data item can then be accessed if we know the location (address), of the memory cell.
 - The address of the variable **ptr** can be determined by the expression **&ptr** where **&** evaluates the address of the operand (i.e., **ptr**).
- The following one helps you to understand representation of a pointer variable.

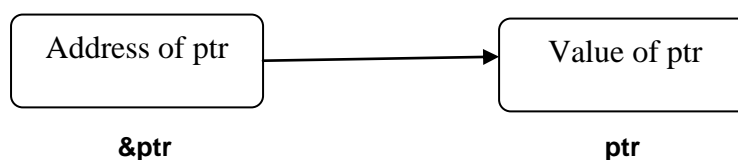


Fig : Representation of a pointer variable.

For example,

```

main( )
{
    int *ptr, i=25;
    ptr = &i;
}
  
```

where

➤ **ptr** is a pointer variable for storing the address of an integer value, hence **ptr** is an integer pointer.

➤ In the above program segment, **ptr** is made to point **i** by storing the address of **i** (i.e., **&i**) in **ptr**.

Ex. Program.1:-

- In following **program** we are using a pointer variable ***ptr**. The pointer variable is initialized by assigning a variable,

```
ptr = &a;
```

➤ In the expression **b = *ptr**, the symbol ***** is placed before a pointer to access the contents of the variable which is assigned to it.

➤ One can be surprised when you see the output of the program particularly the value of the address of **a**, when you execute this program in different computers at different times you will get different values of address which will be allocated to the variable **a** at different times.

Ex.Program 1:

```

/* Program for accessing a variable through a pointer */
/* pointer.c */.

#include<stdio.h>
#include<conio.h>
void main()
{
    int a = 5, b = 10, *ptr;
    clrscr();
    ptr = &a;
    printf("\n Initial values a = %d & b * %d", a, b);
    b = *ptr;
    printf("\n Changed values a = %d & b = %d", a, b);
    printf("\n Value of ptr is %d", ptr);
    printf("\n Value of the address of a is %d", &a);
    printf("\n Value of *ptr is %d", *ptr);
    getch( );
}

```

The program displays the following output:

```

Initial values a = 5 & b = 10
Changed values a = 5 & b = 5
Value of ptr is 4094
Value of the address of a is 4094
Value of *ptr is 5

```

Initialization of Pointer Operators:-

- Like ordinary variables, a pointer variable can also be initialized.
- Static and external (global) pointer variables are initialized with NULL by default.
- Automatic (local) pointer variables can be initialized either with NULL or with the address of some other variable, which is already defined.

➤ For example the statement,

```
ptr = &i;
```

✚ is called as **pointer initialization**, since the address of i is initialized to the pointer variable **ptr**.

✚ A pointer variable can be also initialized as an ordinary variable in declaration itself.

For example,

```
int i, *ptr = &i;
```

✚ is also valid, since **ptr** points to the value stored in the address of **i**.

Ex. Program 2:**Ex.Program 2:**

```
/* Program to find the biggest of N numbers in an array using pointers */
/* big_num.c */

#include<stdio.h>
#include<conio.h>
void main()
{
    int i, j, n, a[25], *ptr;
    clrscr();
    printf("Enter the number of elements : ");
    scanf("%d", &n);
    printf("Enter the array elements : ");
    for( i=0;i<n;i++)
    scanf("%d", &a[i]);
    *ptr = a[0] ;
    for(i=0;i<n;i++)
    {
        if(a[i] > *ptr)
        {
            *ptr = a[i];
        }
    }

    printf("The biggest element in the array is %d", *ptr);
    getch( );
}
```

The program displays the following output:

```
Enter the number of elements : 7
Enter the array elements : 672 -44 96 32 772 -82 55
The biggest element in the array is 772
```

CONCEPT OF STRUCTURES

Structures

- ❖ A structure is a derived type usually representing a collection of variables of same (or) different data types grouped together under a single name.
 - The variables (or) data items in a structure are called as **members** of the structure.
 - A structure may contain one or more integer variables, floating-point variables character variables, arrays, pointers, and even other structures can also be included as members.
 - Structures help to organize data, particularly in large programs, because they allow a group of related variables to be treated as a single unit.
- There are two fundamental differences between an array and a structure.
 - ❑ One is, an array demands a homogeneous data type, i.e., the elements of an array must be of the same data type, where as a structure is a heterogeneous data type, since it can have any data type as its member.
 - ❑ The second difference is that elements in an array are referred by their positions, where as, members, in structure are referred by their unique names.

Declaring a Structures:-

- Structure declarations are somewhat more complex than array declarations since a structure must be defined in terms of its individual members.
 - The general form or the syntax of declaring a structure is

```

struct <structure_name>
{
    data type member1;
    data type member2;
    :
    :
    data type memberN;
};
  
```

In the above declaration,

- ✚ struct is a keyword, followed by an optional user-defined structure name usually referred as a tag, which is used to identify the structure, and then the list of members with its data types.
- ✚ The list of member declarations is enclosed in a pair of flower braces.
- ✚ The closing brace of the structure and the semicolon ends the structure declaration.

- Example for declaring a structure is

```

struct employee
{
    int empno;
    char empname[15];
    float salary;
};

```

Where

- ✚ employee is the name of the structure (i.e., tag).
- ✚ The structure employee contains three members of type int, char and float representing empno, empname and salary respectively.

Defining a Structures:-

- Defining a structure means creating variables to access the members in the structure.
 - Creating a structure variable allocates sufficient memory space to hold all the members of the structure.
 - Structure variables can be created during structure declaration or by explicitly using the structure name.
- The syntax for defining the structure during structure its declaration is,

```

struct <structure name>
{
    data type member1;
    data type member2;
    . . .
    data type memberN;
}structure_variable(s);

```

- An example for defining the structure during declaration itself is,

```

struct employee
{
    int empno;
    char empname[15];
    float salary;
}empl;

```

Where

- ✚ The structure employee declares a variable emp1 of its type.
- ✚ The structure _variable(s) is/are declared like ordinary variables is used to access the members of the structure.
- ✚ More than one structure _variable can also be declared by placing a comma in between the variables.

- The following one shows the memory allocation for the structure employee

empno	2 bytes
empname	15 bytes
salary	4 bytes

Memory occupied by, structure employee.

Program:

```

Ex.Program:

/* structure.c */

#include<stdio.h>
#include<conio.h>
struct employee
{
    int empno;
    char empname[30];
    char dept_name[30];
    float salary;
}emp;

void main( )
{
    struct employee;
    clrscr( );
    printf("Enter empno,empname,dept_name,salary:\n");
    scanf("%d%s%s%f", &emp.empno, &emp.empname,
        &emp.dept_name, &emp.salary);
    printf("\nEmployee no:%d\n", emp.empno);
    printf("Employee name:%s\n", emp.empname);
    printf("Department name:%s\n", emp.dept_name);
    printf("Employee salary:%.f\n", emp.salary);
    getch();
}

```

The program displays the following output:

```

Enter empno,empname,dept_name,salary:
    1 Devankumar  ComputerScience 23600
Employee no: 1
    Employee name: Devankumar
    Department name: ComputerScience
    Employee salary: 23600

```

CONCEPT OF UNIONS***Unions***

- ❖ A union is another compound data type like a structure that may hold objects of different types and sizes with the compiler keeping track of the size.

- Unions provide a way to manipulate different kinds of data in a single area of storage.

- The general form (or) the syntax of defining a union is

```
union <union name>
{
    data type member1;
    data type member1;

    .
    .
    data type memberN;
}union_variable(s);
```

- The syntax of a union is identical to that of a structure except that the keyword struct is replaced with the keyword union.

- An example for declaring a union is

```
union exam
{
    int roll_no;
    char name[15];
    int mark1, mark2, mark3;
}
```

In the above example,

- ✚ union exam has 5 members.
- ✚ The first member is a character array name having 15 characters (i.e., 15 bytes).
- ✚ The second member is of type int that requires 2 bytes for their storage.
- ✚ All the other members mark1, mark2, marks are of type int which requires 2 bytes for their storage.

- The following one shows the memory allocation of members of union exam.

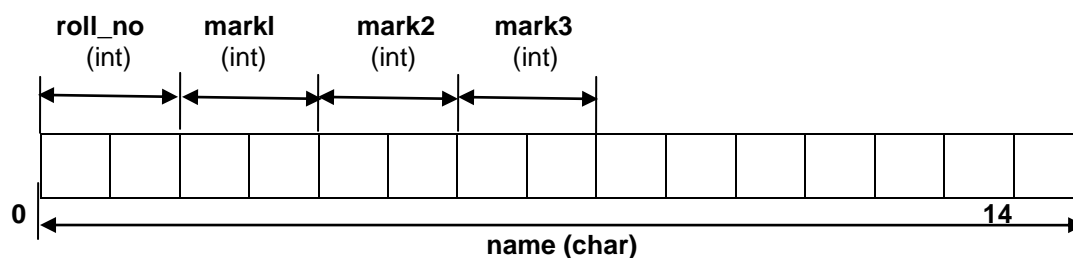


Fig: Memory occupied by union exam

- In the union, all these 5 members are allocated in a common place of memory (i.e., all the members share the same memory location).

- A union shares the memory space instead of wasting storage on variables that are not being used.
- The compiler allocates a piece of storage that is large enough to hold the largest type in the union.
- In the above declaration, the member name requires, 15 characters, which is the maximum of all members, hence a total memory space of 15 bytes is allocated. In case of structures, the total memory space allocated will be 23 (i.e., 15+2+2+2+2) bytes.

Ex. Program:**Ex.Program:**

```

/* union.c */
/* size.c */

union exam1
{
    int roll_no;
    char name [15] ;
    int mark1 , mark2 , mark3 ;
}ul;

struct exam2
{
    int roll_no;
    char name [15] ;
    int mark1 , mark2 , marks ;
}s1;

#include<stdio.h>
#include<conio.h>
void main( )
{
    clrscr();
    printf ("The size of the union is %d\n" , sizeof (ul) );
    printf ("The size of the structure is %d" ,sizeof (s1));
    getch();
}

```

The program displays the following output:

```

The size of the union is 15
The size of the structure is 23

```

LECTURE 14:

INTRODUCTION TO A DATA STRUCTURES***Data Structures******Introduction:-***

- ❖ A Data structure represents the logical relationship that exists between individual elements of data to carry out certain tasks.
(or)
- ❖ A Data structure defines a way of organizing all data items that consider not only the elements stored but also stores the relationship between the elements.
(or)
- ❖ The Data structure as the collection of those elements and all the possible operations which are required for those set of elements. The operations includes inserting, deleting, searching and printing an element. It is a way of representing logical relationship between individual elements.
- To develop a program for an algorithm, we should first decide the steps and select an appropriate data structure for that algorithm. The choice and implementation of a data structure is as important for easier manipulation of data. Therefore, algorithms, and data structures are closely related to each other for developing a program.

TYPES OF DATA STRUCTURES (PRIMARY & SECONDARY DATA STRUCTURES)

Data Structures

- ❖ A Data structure represents the logical relationship that exists between individual elements of data to carry out certain tasks.

(or)

- ❖ A Data structure defines a way of organizing all data items that consider not only the elements stored but also stores the relationship between the elements.
- ❖ Data Structures can be broadly classified as two types, they are

1. Primary Data Structures
2. Secondary Data Structures

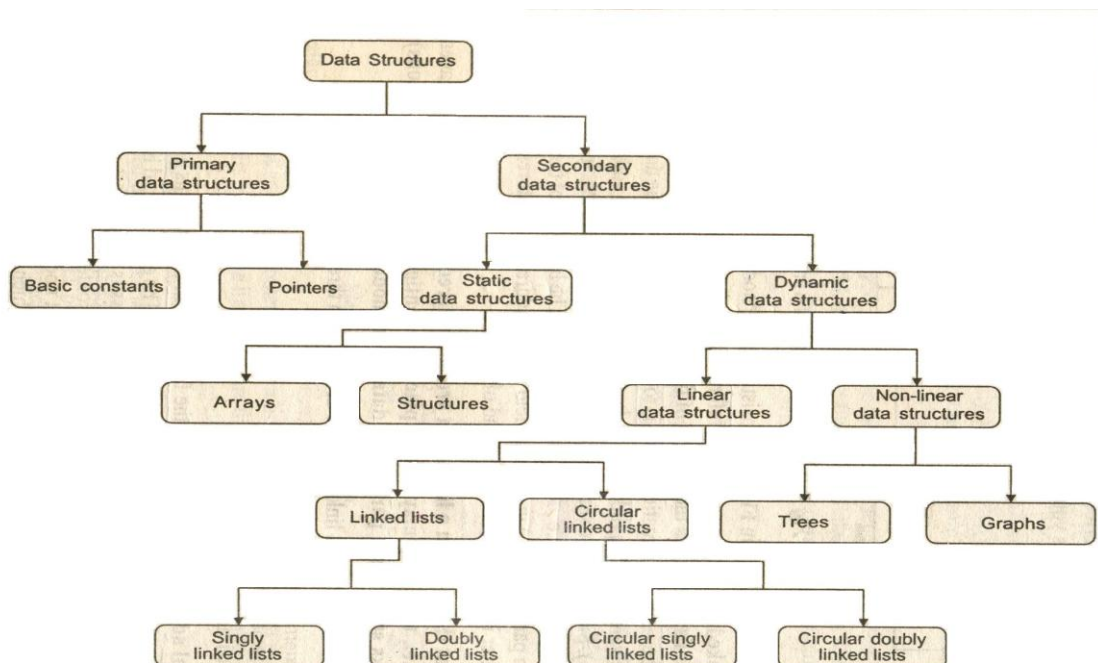
Primary Data Structures:-

- Primary data structures are the basic data structures that directly operate upon the machine instructions.
- They have different representations on different computers.
- All the basic constants (integers, floating-point numbers character constants, string constants) and pointers are considered as primary data structures.

Secondary Data Structures:-

- Secondary data structures are more complicated data structures derived from primary data structures.
- They emphasize on grouping same (or) different data items with relationship between each data item.
 - Secondary data structures can be broadly classified as two types:
 - ✚ Static data structures
 - ✚ Dynamic data structures
 - If a data structure is created using static memory allocation, (i.e., a data structure formed when the number of data items are known in advance) it is known as static data structure (or) fixed size data structure.
 - If a data structure is created, using dynamic memory allocation (i.e., a data structure formed when the number of data items are not known in advance) is known as dynamic data structure (or) variable size data structure.
 - Dynamic data structures can be broadly classified as two types:
 - ✚ Linear data structures
 - ✚ Nonlinear data structures
 - Linear data structures, have a linear relation ship between its adjacent elements.

- Linked lists are examples of linear data structures.
- A linked list is a linear dynamic data structure that can grow and shrink during its execution time.
- A circular linked similar to a linked list except that the first and last nodes are interconnected.
- Non-linear data structures don't have a linear relationship between its adjacent elements.
- In a linear data structure, each node has a link which points to another node, where as in a non-linear data structure, each node may point to several other nodes.
- A **tree** is a non-linear dynamic data structure that may point to one or more nodes at a time.
- A **graph** is similar to tree except that it has no hierarchical relationship between its adjacent elements.

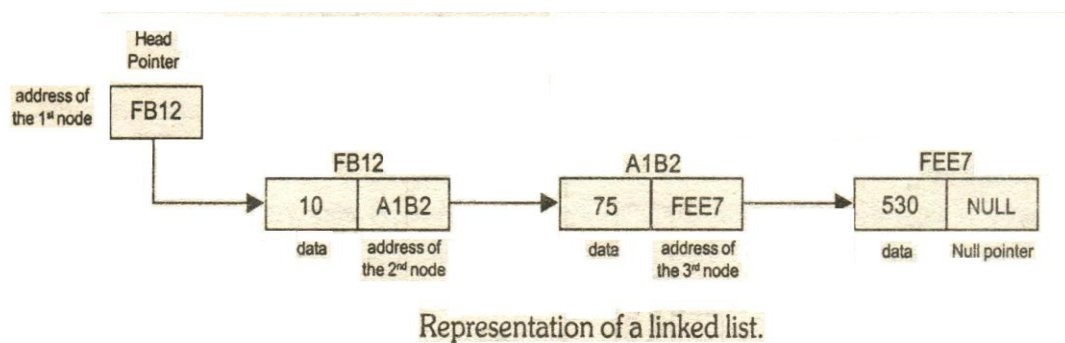


Various types of data structures.

CONCEPT OF A LINKED LISTS & TYPES OF A LINKED LISTS

Linked Lists

- ❖ Linked list (or) list is an ordered collection of elements. Each element in the list is referred as a node. Each node contains two fields namely,
 - Data field
 - Link field
- The data field contains the actual data (i.e., value) of the element to be stored in the list and the link field also referred as the “**next address field**” contains the address of the next node in the list.



- The linked list shown in Fig., consists of three nodes, each with a data field and a link field, A linked list contains a pointer, referred as the **head pointer**, which points to the first node in the list that stores the address of the first node of the list (i.e., FB12).
 - The data field contains the actual information which is to be stored in the list.
 - The data field of the first node stores the value 10.
 - The link field of the first node contains the address of the second node (i.e., A1B2).
 - Similarly, the second node of the list stores the value 75 in the data field and the address of the third node (i.e., FEE7) in the link field.
 - The last node of the list contains only the information part (i.e., 530) in the data field and the address field stores the NULL pointer.
 - This NULL pointer is used to indicate the end of a list.
- The address stored in a linked list are divided into three types namely,
- ❖ External address
 - ❖ Internal address
 - ❖ Null address.
- External address** is the address of the first node in the list. This address is stored in the head pointer which points to the first node in the list.
 - The entire linked list can be accessed only with the help of the head pointer.

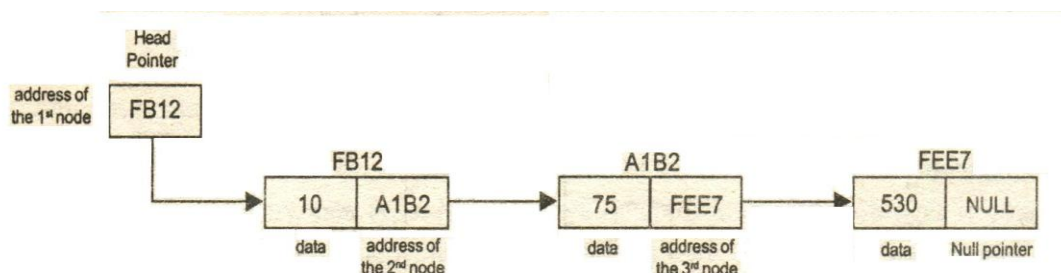
- ❑ **Internal address** is the address stored in each and every node of the linked list except the last node.
- ❑ The content stored in the link field (also referred as next address field) is the address of the next node.
- ❑ **Null address** is the address stored by the NULL pointer of the last node of the list, which indicates the end of the list.

Types of Linked Lists:-

- There are different types of linked lists. They can be classified as,
 - ❖ Singly linked list
 - ❖ Doubly linked list
 - ❖ Circular linked list.

Singly Linked List:-

- The list that we have seen so far is referred to as the **singly linked list**, in which each node has a single link to its next node. This list is also referred as a **linear linked list**.
- The head pointer points to first node in the list and the null pointer is stored in the link field of the last node in the list, which indicates end of list.
- You can traverse (move) in a singly linked list in only one direction (i.e., from head to null in a list). You cannot traverse the list in the reverse direction (i.e., from null to head in a list).



Representation of a linked list.

Fig: Representation of Doubly linked list.

Doubly Linked List:-

- For some applications, especially those where it is necessary to traverse lists in both directions, doubly linked lists work much better than singly linked lists.
 - Doubly linked list is an advanced form of a singly linked list, in which each node contains three fields namely,
 - ❖ Previous address field
 - ❖ Data field
 - ❖ Next address field.
- The previous address field of a node contains address of its previous node. This field is also referred as the **“backward link field”**.
- The data field stores the information part of the node. The next address field contains the address of the next node in the list. This field is also referred as the **“forward link field”**.

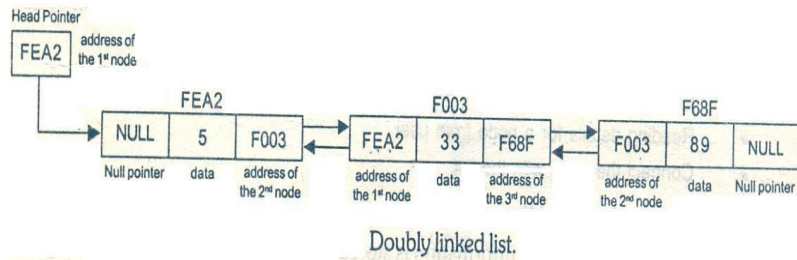


Fig: Representation of Doubly linked list.

Circular Linked list:-

- ❑ Circular linked list is another form of a linked list in which the last node of the list is connected to the first node in the list.
- There are different types circular linked lists. They can be classified as,
 - ❖ Circular singly linked list
 - ❖ Circular doubly linked list

Note:

- ❖ That a circular linked list looks like a cyclic list where there won't be any end-of-list (i.e., there is no NULL pointer).

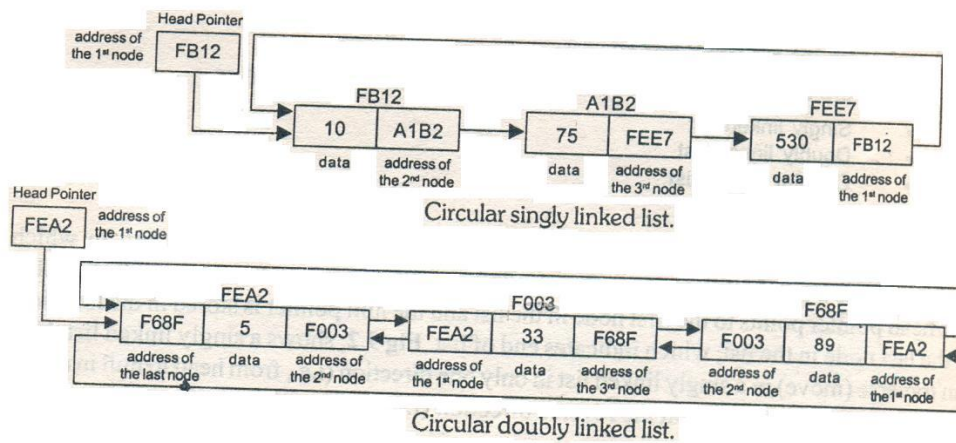


Fig: Representations of Circular singly linked list & Circular doubly linked list.

BASIC OPERATIONS OF A LINKED LISTS

Basic Operations in a Singly Linked List

- ❖ The most commonly used linked list discussed so far is the singly linked list.
 - The basic operations that can be performed on singly linked lists are,
 - ✚ Creation of a list
 - ✚ Insertion of a node
 - ✚ Modification of a node
 - ✚ Deletion of a node
 - ✚ Traversal of a list

Advantages of linked lists:-

- It is not necessary to specify the number of elements in a linked list during its declaration (since memory can be allocated dynamically when a node is added to the list).
- Linked list can grow and shrink in size depending upon the insertion and deletion that occurs in the list.
- Insertions and deletions at any place in a list can be handled easily and efficiently.
- A linked list does not waste any memory space.

Disadvantages of linked lists:-

- Searching a particular element in a list is difficult and time consuming.
- A linked list will use more storage space than an array to store the same number of elements (∵ each element in a list needs additional memory space for storing the address of the next node).

LECTURE 16:

CONCEPT OF STACKS & OPERATIONS ON STACKS (PUSH & POP Operations)***Stacks***

- ❖ A stack is an ordered collection of elements in which insertions and deletions, are restricted to one end.
 - The end from which elements are added and/or removed is referred to top of the stacks.
 - Stacks are also referred as "**piles**" and "**push-down lists**".
 - The first element placed in the stack will be at the bottom of the stack.
 - The last element placed in the stack will be at the bottom of the stack.
 - The last element placed in the stack will be at the top of the stack.
 - The last element added to stack is the first element to be removed. Hence stacks are referred to as **Last-In-First-Out (LIFO)** lists.
 - A stack is referenced via a pointer to the top element of the stack referred as "**top pointer**".
 - The top pointer keeps track of the top element in the stack.
 - Initially, when the stack is empty, the top pointer has a value zero and when the stack contains a singly element, the top pointer has a value one and so on.

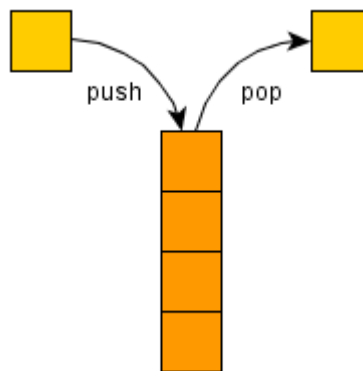


Fig: shows the representation of a stack.

- The primary operations that can be carried on a stack are insertions and deletions.
- These operations on a stack are referred to as PUSH and POP operations respectively.
- A PUSH operation adds a new element to the stack. Each time a new element is inserted in the stack, the top pointer is incremented by one before the element is placed on the stack.
- A POP operation deletes the top most element in the stack. Each time an element is removed from the stack the top pointer is decremented by one.

Representation of Stacks Using Arrays:-

- One of the simplest ways of representing a stack is by means of a singly dimensional array.
- Both stacks and arrays are ordered collection of elements, but an array and attack are two different things.
- The number of elements in the arrays fixed and it is not possible to change the number of elements, but the size of the stack varies continuously when elements are pushed and popped.
- The stack is stored in a part of the array, so an array can be declared large enough to hold the maximum number of elements of the stack.
- During execution of a program, the stack size can be varied within the space reserved for it. One end of the array (i.e., bottom of the stack) is fixed and the other end of the array (i.e., top of the stack) is constantly changed depending upon the elements pushed and/or popped in the stack.
- Care must be taken in handling extreme cases, like not to push an element into a “**fully occupied stack**” and not to pop an element from an “**empty stack**”
 - To represent a stack using arrays, we need,
 - ❖ An array to hold the elements of the stack, which can be of any data type such as int, char, float, etc.,
 - ❖ An integer variable to indicate the top position of the stack within the array.

PUSH Operation:-

- PUSH is an operation used to add an element into stack.
- The PUSH operation of a stack is implemented using arrays.
- When implementing the PUSH operation, “**overflow condition**” of a stack is to be checked (i.e., you have to check whether the stack is full or not).
- If the size of the stack is defined as 5, then it is possible to insert (i.e., add) only 5 elements into the stack.
- It is not possible to add any more elements to the stack, since there is no space to accommodate the elements in the array.
- The following procedure helps you to understand things better.
 - Create a function PUSH() with one argument, the ELEMENT to be added.
 - Assign the new element in the array by incrementing the TOP variable by 1.

POP operation:-

- POP is an operation used to remove an element from the TOP of the stack.
- POP operation of a stack is implemented using arrays.
- When implementing the POP operation, “**underflow condition**” of a stack is to be checked (i.e., You have to check whether the stack is empty or not).

- The user should not POP an element from an empty stack. This type of an attempt is illegal and should be avoided.
- If such an attempt is made, the user should be informed of the underflow condition.

➤ The following procedure can help you to understand things better.

- Create a function POP() with one argument, the address of the element to store the popped value.
- Decrement the TOP variable by 1.

Empty Stack:-

- If stack contains no elements, it is referred as an empty stack.
- If elements to be popped from a stack if you have to check whether the stack is empty or not, since elements cannot be popped *from* an empty stack nor contents cannot be displayed from an empty stack.
- To check whether the stack is empty or not,
 - We are using a variable top:
 - Assume that you are initializing the variable top to -1.
 - If the variable top is, -1 during execution the output message "stack underflow on POP" is displayed. This condition is referred as empty condition of a stack.

Fully Occupied Stack:-

- If a stack contains elements equal to its size (i.e., MAX), we say that the stack is full.
- If an element is entered into a stack, you have to check whether the stack is full or not, since elements cannot be pushed into a stack if it is fully occupied.
- To check whether the stack is full or not, we are using a variable top.
- We are checking whether the variable top is equal to the maximum size of the array minus one (Since an array starts from zero, if the size of the array is 5, and if the stack is full the last element of the stack is stored in the array location 4).
- If the condition is satisfied the output message, "Stack Overflow on PUSH" is displayed. This condition is referred as fully occupied stack.

Applications of Stack:-

- Some important applications using stacks are
 - Towers of Hanoi
 - Reversing a string
 - Evaluation of arithmetic expressions.

EVALUATION OF ARITHMETIC EXPRESSION (Notations: Prefix, Infix, Postfix)

Evaluation of Arithmetic Expressions

- An-expression consist of two components namely **operands** and **operators**.
 - Operators indicate the operation to be carried out on operands.
 - There two kinds of operators used evaluating expressions, namely unary and binary operator.
 - Unary operators require only one operand to carryout its intended operation whereas binary operators require two operands to carry out_its intended operation.
 - Most operators used are binary in nature.
 - Computers solve arithmetic expressions by restructuring them so the order of each calculation is embedded in the expression. Once converted, an expression can then be solved in one pass.
- There are three ways of representing expressions in computers. They are,
- Infix notation
 - Prefix notation
 - Postfix notation
- The prefixes of the notations "**in**", "**pre**", and "**post**" refers to the position of the operands with respect to the operators.
 - All the three ways mentioned above, uses the operator and operand in different ways in evaluating an expression.
 - Table shows the different ways of representing an expression.

Notation	Arithmetic expression
INFIX	Operand OPERATOR Operand
PREFIX	OPERATOR Operand Operand
POSTFIX	Operand Operand OPERATOR

The Infix Notation:-

- The normal way of expressing mathematical expressions is called as "***infix notation***".
- In this form of expressing an arithmetic expression the operator comes in between its operands.
- For example, an expression to add numbers "a" and "b" is written in infix form as

$$(a + b)$$

- Note that the operator "+" is written in between the operands "a" and "b".
- We are accustomed to this type of infix notations.
- However, algorithmically, postfix and prefix notations are easier to evaluate than infix notation.
- Hence, let us discuss the conversion of infix to postfix notation and infix to prefix notations.

Advantages of infix notations:

- ❑ It is the mathematical way of representing the expression.
- ❑ It's easier to see visually which operation is done from first to last.

The Prefix Notation:-

- Prefix notation referred, as "**polish notation**" is a way of writing algebraic expressions without the use of parenthesis or rules of operator precedence.
- In this form of expressing an arithmetic expression the operator is written before its operands.
 - For example, an expression to add numbers "a" and "b" is written in prefix form as
(+ a b)
 - ❑ Note that the operator "+" is written before the operands "a" and "b".

The Postfix Notation:-

- Postfix notation also referred, as "**Suffix form (or) Reverse Polish Notation (RPN)**" is without the use of parentheses or rules of operator precedence.
- In this form of expressing an arithmetic expression the operator is written after its operands.
 - For example, an expression to add numbers "a" and "b" is written in postfix form as
(a b +)
 - ❑ Note that the operator "+" is written after the operands "a" and "b".

Advantages of postfix notations:-

- ❖ You need not worry about the rules of precedence.
- ❖ You need not worry about the rules for right to left associativity.
- ❖ You need not need parenthesis to override the above rules.

CONVERSION OF NOTATIONS

Conversions of Notations

- The main problem in evaluating an expression is to decide the order in which the operations are carried out.
- The order of evaluation of the expression **(a+b)** is quite simple, where as if the expression is complex, we require knowledge of the operators, their precedence to specify the order of evaluation of the expression.
- To fix the order of evaluation for an expression, we assign priority for operators.
- The operators with the highest priority will be evaluated first.
- Since we give more importance to binary operators, the most important binary operations according to their order of priority are listed in Table 10.2. Note that by using parenthesis we can override the default precedence of operators.

Priority	Operation (symbol)
1	Exponentiation (T)
2	Multiplication (*), Division (/)
3	Addition (+). Subtraction (-)

Binary operations in their order of priority

- When un parenthesized operators of the same precedence are scanned, the order of evaluation is assumed from right to left (except for exponentiation whose precedence is left to right).
- To every infix expression there corresponds a postfix expression that has the same effect.
- The reverse is not true because, there is an ambiguity.
- The infix expression **x+y+z** can be represented as either **xyz++** (or) **xy+z+** in postfix.
- The reason for the ambiguity is the lack of parenthesis in the infix expression.
- If the infix expression were fully bracketed there would be no ambiguity.
- Thus **(x+y) +z** pairs with **xy+z+** and **x+(y+z)** pairs with **xyz++**.
-

➤ Rules to be followed during infix to postfix conversion:-

- Fully parenthesize the expression starting from left to right (During parenthesizing, the operators having higher precedence are first parenthesized).
- Move the operators one by one to their right, such that each operator replaces their corresponding right parenthesis.
- The part of the expression, which has been converted into postfix, is to be treated as single operand.
- Once the expression is converted into postfix form, remove all parenthesis.

Example 1:

- ❖ Evaluate the infix expression $3+8*4/2-(8-3)$ in postfix notation.

Infix expression:- $3+8*4/2-(8-3)$

Postfix expression:- $3\ 8\ 4\ *\ 2\ /\ +\ 8\ 3\ -\ -$

Example 2:-

- Give the postfix form for the infix expression $x+y * z$.

Solution:

- The order of evaluation of postfix form is

- | | | |
|---|---|--------------------------|
| i. Parenthesize the operands with operator having the highest priority | = | $X + (Y * Z)$ |
| ii. Move the operator inside parenthesis to the right of the operand | = | $X + (YZ *)$ |
| iii. Consider the part of the expression converted to postfix as a single operand | = | $X + A$ [$A = (YZ *)$] |
| iv. Parenthesize the sub-expression | = | $(X + A)$ |
| v. Move the operator inside parenthesis to the right of the operand | = | $(X A +)$ |
| vi. Substitute the value for A | = | $(X (YZ *) +)$ |
| vii. Remove all parenthesis | = | XYZ^*+ |

∴ The required postfix form is XYZ^*+

Example 3:-

- ❖ Give the postfix form for the infix expression $P + Q/R - S$.

Solution:

- The order of evaluation of postfix form is:

- | | | |
|---|---|-----------------------------|
| i. Parenthesize the operands with operator having the highest priority | = | $P + (Q / R) - S$ |
| ii. Move the operator inside parenthesis to the right of the operand | = | $P + (Q R /) - S$ |
| iii. Consider the part of the expression converted to postfix as a single operand | = | $P + A - S$ [$A = (QR/)$] |
| iv. Parenthesize the sub-expression with operator having the highest priority | = | $(P + A) - S$ |

- v. Move the operator inside parenthesis to the right of the operand = $(P A +) - S$
- vi. Consider the part of the expression converted to postfix as a single operand = $B - S$ [$B = (PA +)$]
- vii. Parenthesize the sub-expression = $(B - S)$
- viii. Move the operator inside parenthesis to the right = $(B S -)$
- ix. Substitute the value for B = $((PA+) S -)$
- x. Substitute the value for A = $((P (QR /) +) S -)$
- xi. Remove all parenthesis = $PQR/+S-$

∴ The required postfix form is $PQR/+S-$

CONCEPT INFIX NOTATION TO POSTFIX NOTATION

Converting Infix Expression to Postfix Form

- Evaluation of an expression using computers can be done in different ways.
- Some programmers may try to parse the expression and find out the operators that have the highest priority, calculate them and later evaluate the expression with operators of lower priority.
- But if parentheses and functions are added to the expression it will soon be very complicated to do the calculation.
- An easier way to calculate it is to convert the expression from infix notation to postfix notation.
- This may first look very strange but this is much easier when we need to do calculations on a computer.
 - The following procedure is used to convert infix to postfix expression:
 - ✚ Read the infix string.
 - ✚ Traverse from left to right of the expression.
 - ✚ If an operand is encountered, add to the postfix string.
 - ✚ If the item is an operator push it on the stack, if any of the following conditions are satisfied.
 - The stack is empty.
 - If the precedence of the operator at the top of the stack is of lower priority than the operator being processed.
 - ✚ If all the above condition fails, then pop the operator being processed to the postfix string.
 - ✚ When the infix string is empty, pop the elements of the stack onto the postfix string to get the result.

EVALUATING AN EXPRESSION IN POSTFIX NOTATION

Evaluating an Expression in Postfix Notation

- Evaluating an expression in postfix notation is trivially easy if you use a stack.
- The postfix expression to be evaluated is scanned from left to right.
- Variables or constants are pushed onto the stack.
- When an operator is encountered, the indicated action is performed using the top two elements of the stack, and the result replaces the operands on the stack.

➤ Steps to be noted while evaluating a postfix expression using a stack:-

- Traverse from left to right of the expression.
- If an operand is encountered, push it onto the stack.
- If you see a binary operator, pop two elements from the stack, evaluate those operands with that operator, and push the result back in the stack.
- If you see a unary operator, pop one element from the stack, evaluate those operands with that operator, and push the result back in the stack.
- When the evaluation of the entire expression is over, the only thing left on the stack should be the final result.
- If there are zero or more than 1 operands left on the stack, either your program is inconsistent, or the expression was invalid.

➤ Important points to be noted while evaluating a postfix expression using a stack:-

- When you convert infix expression to and postfix notation, the operands are always in the same order and the operators are probably in a different order.
- The first element you pop off of the stack in an operation, should be evaluated on the right-hand side of the operator.
- For multiplication and addition, order doesn't matter, but for subtraction and division, your answer will be incorrect if you change your operands around.

CONCEPT OF QUEUES & TYPES OF QUEUES

Queues

- ❖ **A queue is an** ordered collection of elements in which insertions are made at one end and deletions are made at the other end.
 - The end at which insertions are made is referred to **as the rear end**, and the end from which deletions are made is referred to as the front end.
 - The first element placed in a queue will be at the start of the queue.
 - In a queue, the first element placed in a queue will be at the start off the queue.
 - In a queue, the first element to be removed.
 - So a queue is sometimes referred to as **First-In-First-Out (FIFO)** lists.
 - Consider five persons waiting in front of a ticket counter in a line for buying their tickets.
 - The person who is standing in front of the line will get the" first ticket, the second person will get the next ticket, and so on.
 - If a new person wants to buy a ticket (say the sixth person) he should stand after the fifth person. These similar operations are carried out in a queue.
 - Queues have many applications in computer systems.
 - Most computers have only a singly processor, so only one user may be served at a time.
 - Entries from other users are placed in a queue.
 - Each entry gradually advances to the front of the queue as users receives their service.
 - Queues are also used to support print spooling.
- The queue shown in **Fig** consists of 5 elements i, 2, 3, 4 and 5.

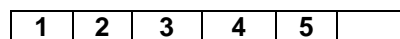


Fig: Representation of a queue

- Where front end is the pointer pointing to the first element in the queue, rear end is the pointer pointing to the last element in the queue.
- Element 1 is the first element of the queue, and 5 is the last element in the queue.
- If you want to delete an element say 3, you have to first delete element 1 and then element 2 and then the element 3.
- The front end is shifted from 1st element 1 to 4.

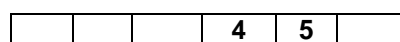


Fig: Queue representation after deletion

- Similarly if you want to add a new element (say 6) it is added after 5 because it is in the rear end.

- After inserting a new element, the rear end is shifted from element 5 to element 6, which is the last position of the queue. Beyond this position, you cannot insert any elements in the queue.

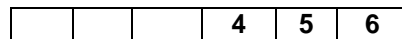


Fig: Queue representation after insertion.

- The primary operations that can be done on a queue are insertions and deletions.
- These operation on a queue are referred as enqueue and dequeue operations respectively.
- An enqueue operation adds a new element in queue.
- This process is carried out by incrementing the rear end and adding a new element at the rear end position.
- A dequeue operation is carried out by incrementing the front end and deleting the first element at the front end position.

Types of Queues:-

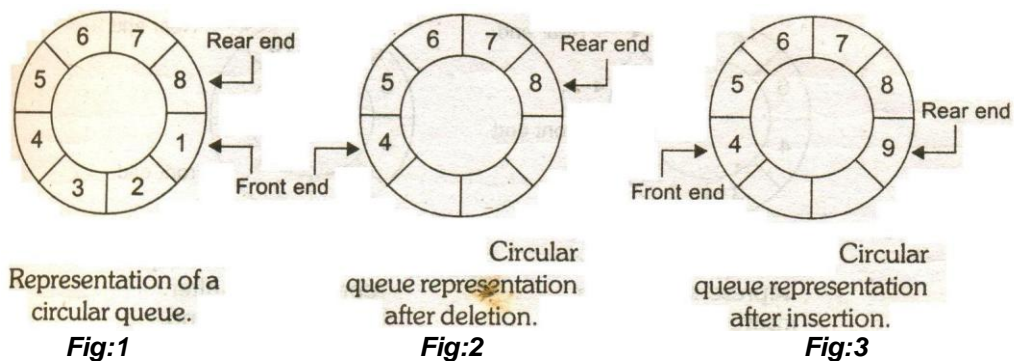
- There are different types of queues. They can be classified as,
 - ❖ Linear Queues
 - ❖ Circular Queues
 - ❖ Deques

Linear Queues:-

- The queue that we have seen so far is referred to as the linear queue.
- The queue has two ends, the front end and the rear end.
- The rearend is where we insert elements and front is where we delete elements.
- You can traverse (move) in a linear queue in only one direction (i.e., from front to rear).
- If the front pointer is in the first position and the rear pointer is in the last position, the queue is said to be fully occupied.
- Initially the front and rear ends are at same positions (i.e.,-1).
- When you insert elements the rear pointer moves one by one (where as the front pointer doesn't change) until the last index position is reached.
- Beyond this you cannot insert the data irrespective of the position of the front pointer. This is the main disadvantage of linear queues, which is overcome in circular queues.
- When you delete the elements the front pointer moves one by one (where as the rear pointer doesn't change) until the rear pointer is reached.
- If the front pointer reaches the rear pointer, both their positions are initialized to -1, and the queue is said to be empty.

Circular Queues:-

- Circular queue is another form of a linear queue in which the last position is connected to the first position of the list.
- The circular queue is similar to linear queue has two ends, the frontend and the rear end.
- The rear end is where we insert elements and front end is where we delete elements.
- You can traverse (move) in a circular queue in only one direction (i.e., from front to rear).
- Initially the front and rear ends are at same positions (i.e., -1).
- When you insert elements the rear pointer moves one by one (where as the front pointer doesn't change) until the front end is reached.
- If the next position of the rear is front, the queue is said to be fully occupied.
- Beyond this you cannot insert any data. But if you delete any data, you cannot insert the data accordingly.
- When you delete the elements the front pointer moves one by one (where as the rear pointer doesn't change) until the rear pointer is reached.
- If the front pointer reaches the rear pointer, both their positions are initialized to -1, and the queue is said to be empty.

**Deque:-**

- Deque (Double-ended queue) is another form of a queue in which insertions and deletions are made at both the front and rear ends of the queue.
 - There are two variations of a deque, namely,
 - Input restricted deque
 - Output restricted deque.
- The input restricted deque allows insertion at one end (it can be either front or rear) only.
- The output restricted deque allows deletion at one end (it can be either front or rear) only.
 - The different types deques are,
 - Linear Deque
 - Circular Deque
 - Linear deque is similar to a linear queue except the following conditions,

- The insertions and deletions are made at both the front and rear ends of the deque.
- If the front end is in the first position, you cannot insert the data at front end.
- If the rear end is in the, last position, you cannot insert at data at rear end.



Fig: Representation of a deque.

➤ Circular deque is similar to a circular queue except the following conditions,

- The insertions and deletions are made at both the front and rear ends of the deque.
- Irrespective of the positions of the front and rear end, you can insert and delete data.

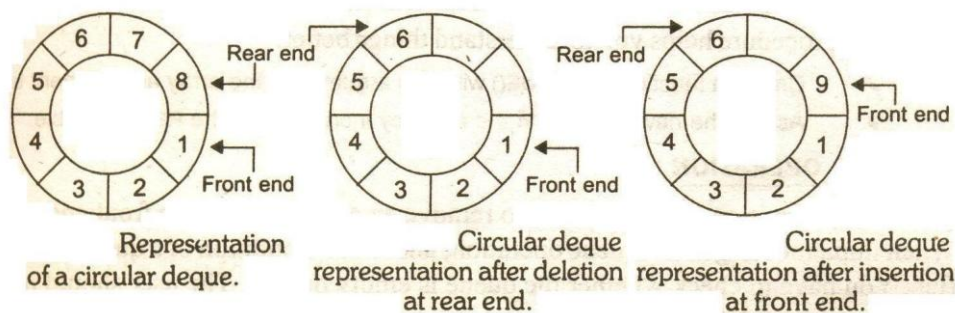


Fig:1

Fig:2

Fig:3

Representation of Linear Queues using Arrays:

- One of the simplest ways of representing a queue is by means of a single dimensional array.
- Both queues and arrays are ordered collection of elements.
 - However, an array and a queue are two different things.
 - The number of elements in the array is fixed and it is not possible to change the number of elements stored in the array.
 - But the size of a queue is constantly changed when elements are enqueued and dequeued.
 - The queue is stored in a part of the array, so an array can be declared large enough to hold the maximum number of elements of the queue.
 - During execution of a program, the queue size can be varied within the space reserved for it. One end of the array (i.e., bottom of the queue) is referred as the rear end and the other end of the array (i.e., top of the queue) is referred as the front end is constantly changed depending upon the elements enqueued (or) dequeued in the queue.
 - Care must be taken in handling extreme cases, like not to insert an element into a fully occupied queue and not to delete an element from an empty queue.

OPERATIONS ON A QUEUE (ENQUEUE & DEQUEUE Operations)

Operations on Queue

- The following are two operations performed on queue, they are

1. Enqueue
2. Dequeue

Enqueue Operation:-

- Enqueue is an operation used to add a new element in to a queue at the rear end.
- When implementing the enqueue operation overflow condition of the queue is to be checked. (Since we cannot insert any data if the rear end is at the last position of the queue).
 - The following procedure helps you to understand things better.
 - ✚ Create a function ENQUEUEQ with two arguments, the array and element to be added.
 - ✚ Assign the new element in the array by incrementing the REAR variable.

Dequeue Operation:-

- Dequeue is an operation used to remove an element from the front end of the queue.
- When implementing the dequeue operation, underflow condition of a queue is to be checked (i.e., You have to check whether the queue is empty or not).
- The user should not delete an element from an empty queue.
- This type of an attempt is illegal and should be avoided.
- If such an attempt is made, the user should be informed of the underflow condition.
 - The following, procedure can help you to understand things better.
 - ✚ Create a function DEQUEUE() with two arguments, the array and the address of the element to store the dequeued value.
 - ✚ Increment the FRONT variable by 1.

Empty Queue:-

- The output restricted deque allows deletion at one end (it can be either front or rear) only.
- If a queue contains no elements, it is referred as an empty queue.
- If an element is to be dequeued from a queue, you have to check whether the queue is empty or not, since elements cannot be dequeued from an empty queue nor contents cannot be displayed from an empty queue.
- To check whether the queue is empty or not the following program segment

Fully Occupied Queue:-

- If a queue contains elements equal to its size (i.e., QSIZE), we say that the queue is full.
- If an element is to be inserted into a queue, you have to check whether the queue is full or not, since elements cannot be inserted into a queue when it is fully occupied.

