LECTURE NOTES

FOR

Problem Solving Through Programming (in C)

I YEAR I SEMESTER B.TECH

(COMPUTER SCIENCE AND ENGINEERING)

2018-19

Department of Computer Science and Engineering

ACE ENGINEERING COLLEGE

(NBA Accredited B.Tech Courses: EEE, ECE, CSE)

(Affiliated to Jawaharlal Nehru Technological University, Hyderabad, Telangana)

Ankushapur (V), Ghatkesar (M), R.R.Dist - 501 301

UNIT – I

Introduction to Computers:

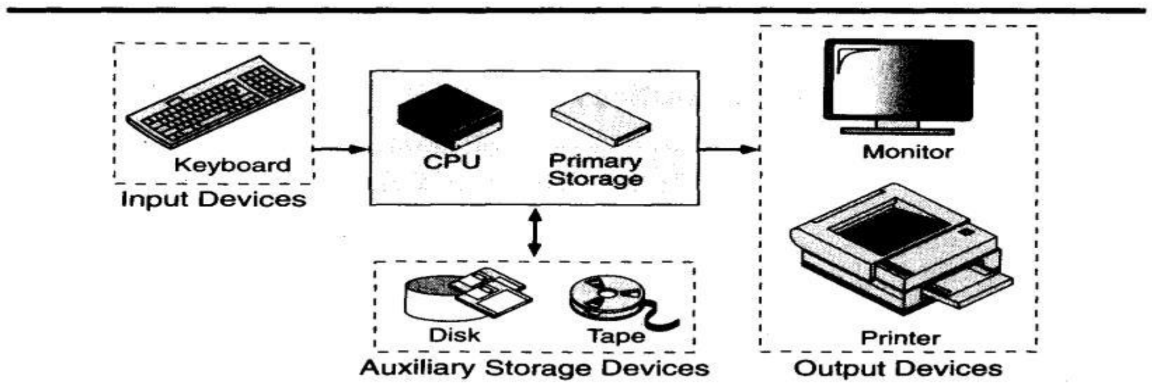
Computer Systems:

- Computer is a machine made of Electronic devices (switch, capacitor, transistor which known 0’s and 1’s or on and off) that enable user to enter data (Input), process it (CPU), and store it in a computer memory.
- A computer is a system made of two major components:
- Hardware and software.
- Hardware is the physical equipment which can be touch and feel.
- Software is the collection of programs (instructions) that perform specific task and allow the hardware to do its job.

Fig1: Different types of computer component

Computer Hardware

- The hardware component of the computer system consists of five parts:
- Input devices,
- Central processing unit (CPU)
- Primary storage,
- Output devices
- Auxiliary storage devices.



Basic Hardware Components

Input device:

Data or instruction are entered into the computer with the help of input devices.
Ex: keyboard, Mouse, Scanner.

Central processing unit (CPU) :

CPU is a computer brain use to perform calculation and other operation.

Output device :

The result given by the computer after processing data is called as output. the output device shows or plays the result after the input has been proceed.
Ex: Monitor, Printer, Speaker are output devices .

Computer Memory:
Computer memory is any physical device capable of storing information temporarily or permanently.

Fig2: Types of Memory

Primary Memory

- Primary memory is computer memory that a computer accesses directly.
- Primary memory is a volatile storage mechanism.
- It may be random access memory (RAM), cache memory or data buses.
- primarily associated with RAM.
- Primary memory is considered faster than secondary memory

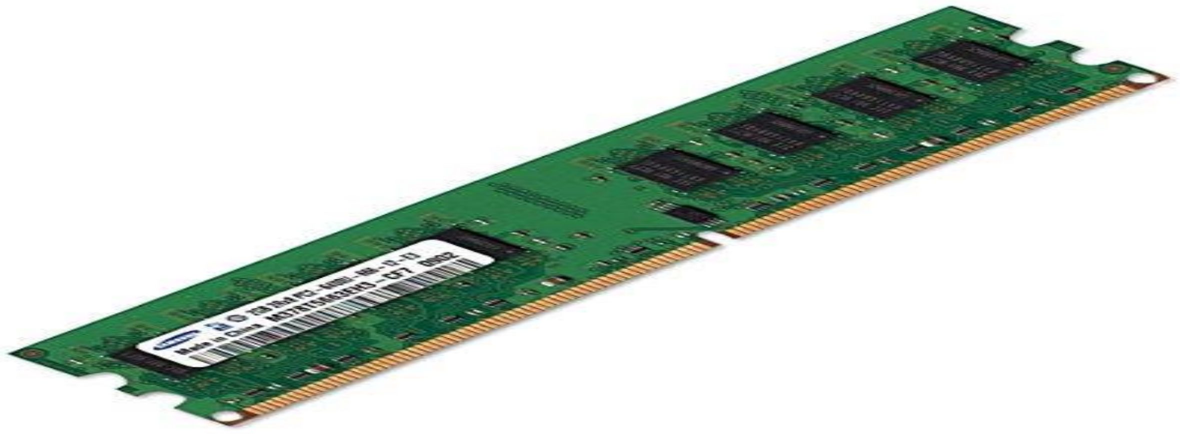
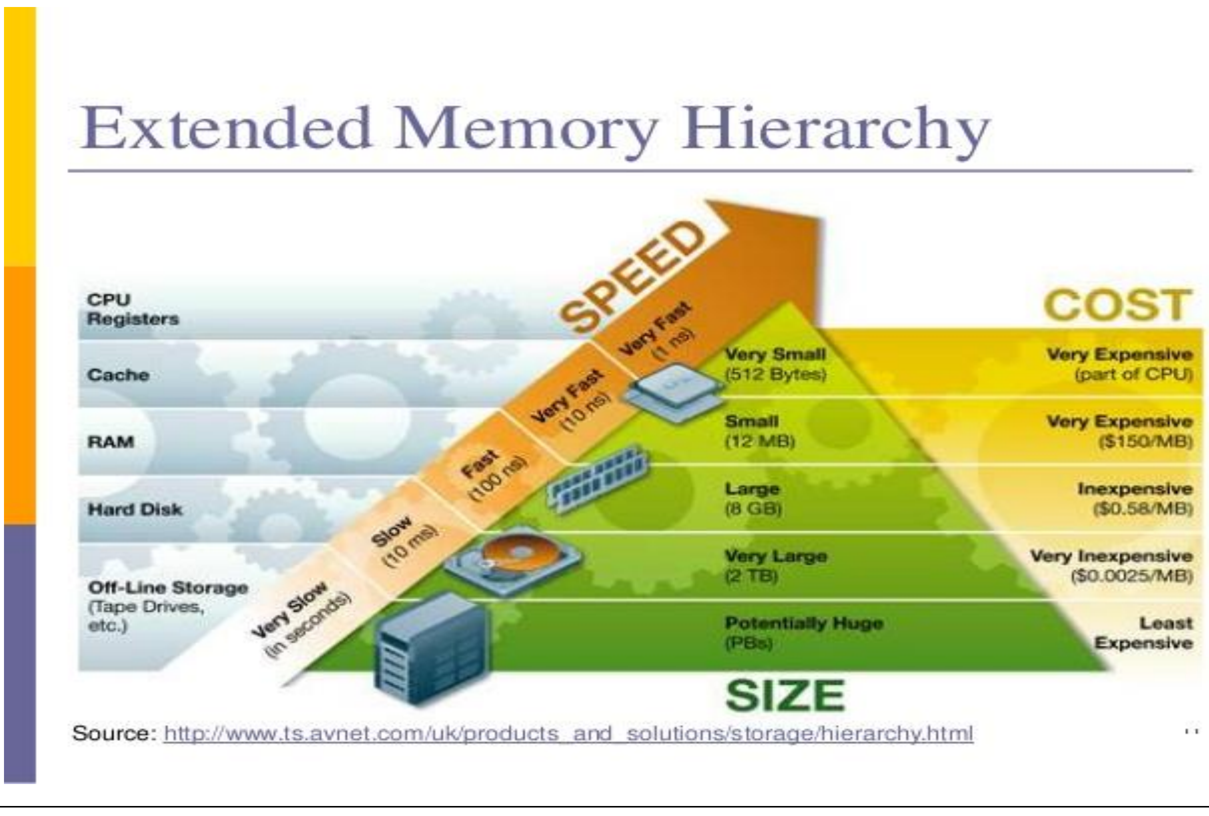


Fig3: Ram

Secondary Storage and Disk

- Secondary storage devices are those devices whose memory is non volatile, meaning, the stored data will be in computer even if the system is turned off.
- Here are a few secondary storage. **Hard Disk, CD RAM,DVD RAM, Pen Drive**
- Secondary storage is also called auxiliary storage.
- Secondary storage is less expensive when compared to primary memory like RAMs.
- The speed of the secondary storage is also lesser than that of primary storage.
- Hence, the data which is less frequently accessed is kept in the secondary storage.



Processor

- A processor is an integrated electronic circuit that performs the calculations that run a computer.
- A processor performs arithmetical, logical, input/output (I/O) and other basic instructions that are passed from an operating system (OS).

Computer Software

- Computer software is divided in to two broad categories: system software and application software.

System software:

- System software manages the computer resources .It provides the interface between the hardware and the users.

- System software consists of programs that manage the hardware resources of a computer and perform required information processing tasks.

- These programs are divided into three classes: the operating system, system support, and system development.

- The operating system provides services such as a user interface, file and database access, and interfaces to communication systems such as Internet protocols.

- System support software provides system utilities and other operating services. Examples of system utilities are sort programs and disk format programs.

- The language translators converts user programs into machine language for execution, debugging tools to ensure that the programs are error free and computer –assisted software engineering (CASE) systems.

Application software

- Application software, on the other hand is directly responsible for helping users solve their problems.

- Application software is classified into two classes:

- **General-purpose software** : is purchased from a software developer and can be used for more than one application and application –specific software.
  Ex: MS-Office, Adobe Photoshop, Adobe Reader

- **specific purpose software**: use only for specific purpose.
  Ex: Library software, Banking software, IRTC software.

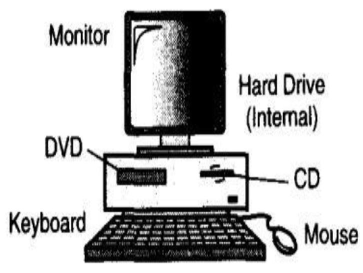
Fig 4: The relationship between system and application software

Computing Environments:

- In the early days of computers, there was only one environment: the main frame computer hidden in a central computing department. With the advent of mini computers and personal computers, the environment changed, resulting in computers on virtually every desktop.

1 Personal Computing Environment

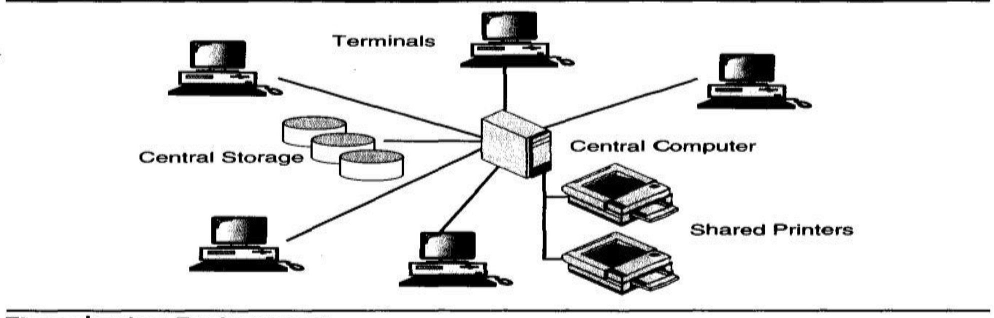
- In 1971, Marcian E.Hoff, working for Intel, combined the basic elements of the central processing unit into the microprocessor.
- The first computer on a chip was the Intel 4004. If we are using a personal computer, all of the computer hardware components are tied together in our personal computer (PC).



Personal Computing Environment

2 Time-Sharing Environment:

- Employees in large companies often work in time-sharing environment.
- Many users are connected to one or more computers.
- These computers may be minicomputers or central mainframes.
- The terminals they use are often nonprogrammable.
- In the time-sharing the output devices and auxiliary storage devices are shared by all of the users.
- A typical college lab in which a minicomputer is shared is shared by many students is shown in figure:

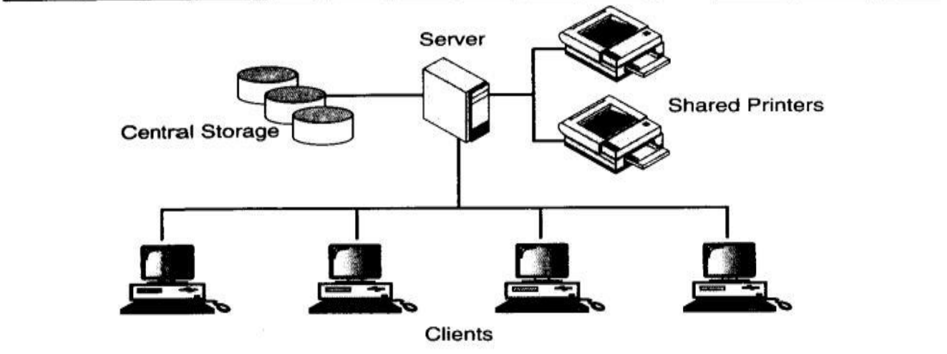


Time-sharing Environment

In the time-sharing environment, all computing must be done by the central computer. The central computer has many duties: It must control the shared resources; it must manage the shared data and printing and it must do the computing.

3 Client/Server Environment

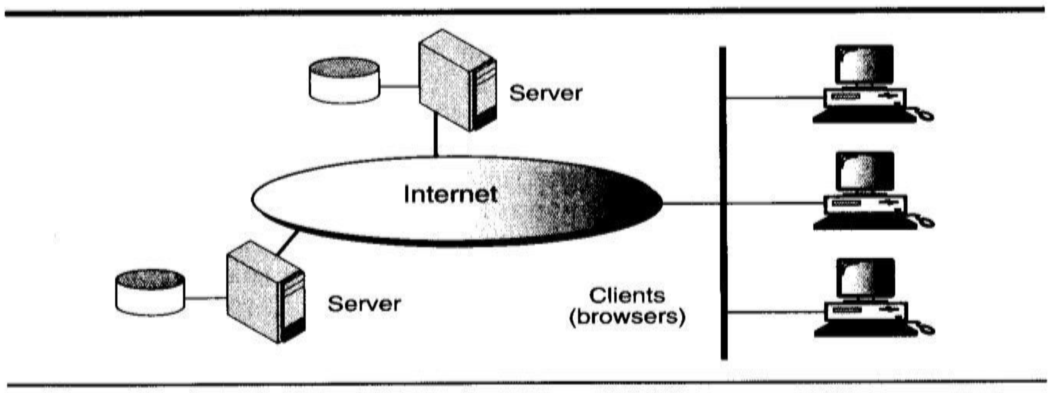
- In the client-server environment, the users' micro computers or workstations are called the client.
- The central computer, which may be a powerful microcomputer, minicomputer, or central mainframe system, is known as the server.
- work is shared between the users' computers and the central computer, hence response time and monitor display are faster and the users are more productive as compare to Time sharing environment.



The Client/Server Environment

4 Distributed Computing

- Distributed Computing environment provides a seamless integration of computing functions between different servers and client's .
- The internet provides connectivity to different servers throughout the world.
- For example eBay uses several computers to provide its auction services. This environment provides a reliable, scalable, and highly available network.



Distributed Computing

Computer Languages:

- To write a program for a computer, we must use a computer language. Over the years computer languages have evolved from machine languages to natural languages.

1940's	Machine level Languages
1950's	Symbolic Languages
1960's	High-Level Languages

Machine Languages

- In the earliest days of computers, the only programming languages available were machine languages. Each computer has its own machine language, which is made of streams of 0's and 1's.

- Instructions in machine language must be in streams of 0's and 1's because the internal circuits of a computer are made of switches transistors and other electronic devices that can be in one of two states: off or on. The off state is represented by 0 , the on state is represented by 1.

The only language understood by computer hardware is machine language.

Symbolic Languages:

- In early 1950's Admiral Grace Hopper, A mathematician and naval officer developed the concept of a special computer program that would convert programs into machine language.

- The early programming languages simply mirror to the machine languages using symbols of mnemonics to represent the various machine language instructions because they used symbols, these languages were known as symbolic languages.

- Computer does not understand symbolic language it must be translated to the machine language.

- A special program called assembler translates symbolic code into machine language. Because symbolic languages had to be assembled into machine language they soon became known as assembly languages.

- Symbolic language uses symbols or mnemonics to represent the various, machine language instructions.

High Level Languages:

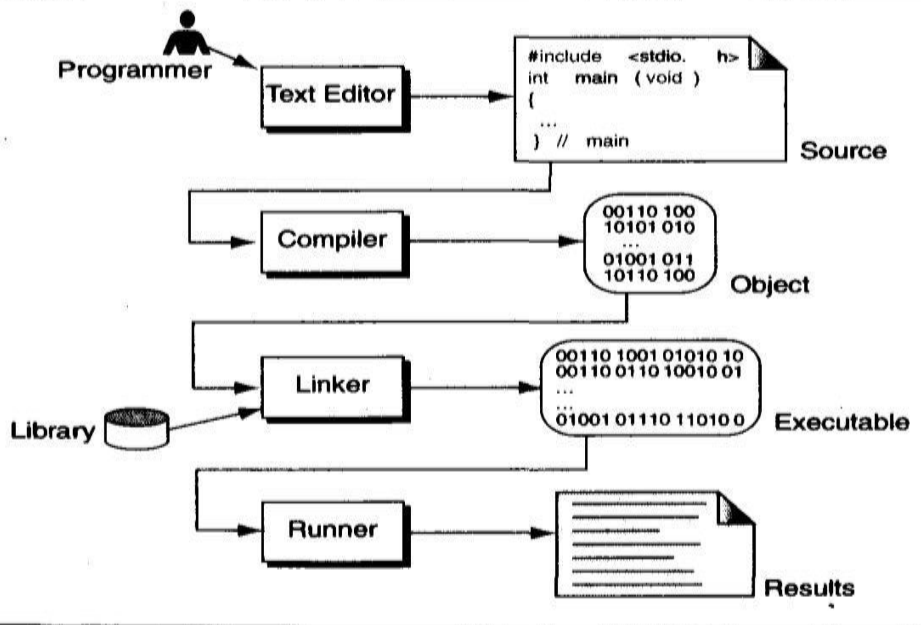
- High level languages are portable to many different computers, allowing the programmer to concentrate on the application problem at hand rather than the intricacies of the computer.
- High-level languages are designed to relieve the programmer from the details of the assembly language.
- High level languages share one thing with symbolic languages; they must be converted into machine language.
- The process of converting them is known as compilation.

The first widely used high-level languages, **FORTRAN (FORmula TRANslation)** was created by John Backus and an IBM team in 1957;it is still widely used today in scientific and engineering applications. After FORTRAN was **COBOL (Common Business-Oriented Language)**. Admiral Hopper was played a key role in the development of the COBOL Business language. C is a high-level language used for system software and new application code.

Creating and Running Programs:

Computer hardware understands a program only if it is coded in its machine language. It is the job of the programmer to write and test the program .There are four steps in this process:

1. Writing and Editing the program
2. Compiling the program
3. Linking the program with the required library modules
4. Executing the program.



Building a C Program

Writing and Editing Programs

- The software used to write programs is known as a text editor.
- A text editor helps user to enter, change, and store character data.
- The main difference between text processing and program writing is that programs are written using lines of code, while most text processing is done with character and lines.
- After writing a program, we save our file to disk. This file will be input to the compiler; it is known as a source file.

Compiling Programs:

- The code in a source file stored on the disk must be translated into machine language; this is the job of the compiler.
- The 'c' compiler is two separate programs. The preprocessor and the translator.
- The code generated after compilation is called object code.

The preprocessor reads the source code and prepares it for the translator. While preparing the code, it scans for special instructions known as preprocessor commands.

After the preprocessor has prepared the code for compilation, the translator convert the program into machine language and generate the object code that is,not executable because it does not have the required C and other functions included.

Linking Programs:

- A C program is made up of many functions.
- Function can be user defined or predefined
- Predefined function, such as input/output function and mathematical library functions that exist elsewhere and must be attached to our program.
- The linker assembles all of these functions code to object code and then generate final executable program.

Executing Programs:

- Once program has been linked, it is ready for execution.
- To execute a program we use an operating system command, such as run,
- OS use the program called loader to load program from secondary memory into primary memory to execute the program

In a typical program execution, it reads data for processing, either from the user or from a file. After the program processes the data, it prepares the output. At output can be to the user's monitor or to a file. When the program has finished its job, it tells the operating system, which then removes the program from memory.

Interpreter Vs Compiler

Interpreter	Compiler
Translates program one statement at a time.	Scans the entire program and translates it as a whole into machine code.
It takes less amount of time to analyze the source code but the overall execution time is slower.	It takes large amount of time to analyze the source code but the overall execution time is comparatively faster.
No intermediate object code is generated, hence are memory efficient.	Generates intermediate object code which further requires linking, hence requires more memory.
Continues translating the program until the first error is met, in which case it stops. Hence debugging is easy.	It generates the error message only after scanning the whole program. Hence debugging is comparatively hard.
Programming language like Python, Ruby use interpreters.	Programming language like C, C++ use compilers.

Errors in C

- Error is an illegal operation performed by the user which results in abnormal working of the program.
- Programming errors often remain undetected until the program is compiled or executed.
- Some of the errors restrict the program from getting compiled or executed. Thus errors should be removed before compiling and executing.

Types of Error

1. **Syntax errors:**
   - Errors that occur when you violate the rules of writing C syntax are known as syntax errors.
   - This compiler error indicates something that must be fixed before the code can be compiled.
   - All these errors are detected by compiler and thus are known as compile-time errors.

- Most frequent syntax errors are:
  Missing Parenthesis (})
  Printing the value of variable without declaring it
  Missing semicolon like this:

2. Run-time Errors :

- Errors which occur during program execution(run-time) after successful compilation are called run-time errors.
- One of the most common run-time error is division by zero also known as Division error.
- These types of error are hard to find as the compiler doesn't point to the line at which the error occurs.

3. Logical Errors :

- On compilation and execution of a program, desired output is not obtained when certain input values are given.
- These types of errors which provide incorrect output but appears to be error free are called logical errors.
- These are one of the most common errors done by beginners of programming.

ALGORITHM:

- An algorithm was developed by an Arab mathematician Abdullah Muhammad bin Musa al-Khwarizmi.
- It is step-by-step approach to solve a given problem.
- **Input**: There are more quantities that are supplied.
- **Output**: At least one quantity is produced
- **Definiteness**: Each instruction of the algorithm should be clear and unambiguous.
- **Finiteness**: The process should be terminated after a finite number of steps.
- **Effectiveness**: Every instruction must be basic enough to be carried out theoretically or by using paper and pencil

Example: Algorithm/pseudo code to add two numbers

```
Step 1: Start
Step 2: Read the two numbers in to a,b
Step 3: c=a+b

Step 4: write/print c
Step 5: Stop.
```

Write an algorithm for converting temperature from degree Celsius to Fahrenheit

2. Write an algorithm for calculating Kinetic Energy. ($E=\frac{1}{2} mv^2$)

PSEUDOCODE:

- Pseudo code is an English like language.that helps programmers develop algorithms.
- Pseudo code is a nonprogrammable.
- it is convenient and user friendly .

FLOW CHART:

A Flow chart is a Graphical representation of an Algorithm or a portion of an Algorithm. Flow charts are drawn using certain special purpose

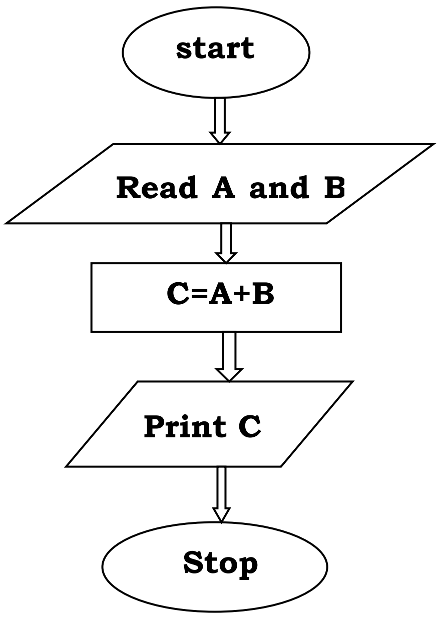
symbols such as Rectangles, Diamonds, Ovals and small circles. These symbols are connected by arrows called flow lines.

(Or)

The diagrammatic representation of way to solve the given problem is called flow chart.

The following are the most common symbols used in drawing flowcharts:

Flowchart for Adding two numbers.



1 Draw a flowchart for converting temperature from degree Celsius to Fahrenheit

3. Draw a flowchart for calculating kinetic Energy. ($E=½ mv^2$)

INTRODUCTION TO 'C' LANGUAGE:

- C language facilitates a very efficient approach to the development and implementation of computer programs. The History of C started in **1972** at the Bell Laboratories, USA where **Dennis M. Ritchie** proposed this language.
- In 1983 the American National Standards Institute (ANSI) established committee whose goal was to produce "an unambiguous and machine independent definition of the language C "while still retaining its spirit.
- C is the programming language most frequently associated with UNIX. Since the 1970s, the bulk of the UNIX operating system and its applications have been written in C.
- Because the C language does not directly rely on any specific hardware architecture, UNIX was one of the first portable operating systems. It appears that there will be yet another ANSI C standard officially dated 1999 or in the early 2000 years; it is currently known as "C9X."

Features of C programming language:

- ❖ Reliability
- ❖ Portability
- ❖ Flexibility
- ❖ Interactivity

- Modularity
- Efficiency and Effectiveness

Uses of C programming language:

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

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

Structure of C program

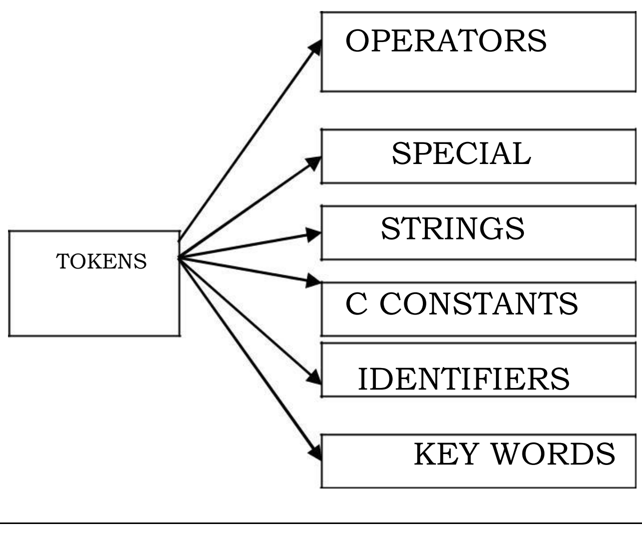
- Structure of C program is defined by set of rules called protocol, to be followed by programmer while writing C program.
- All C programs are having sections/parts which are mentioned below.
    - Documentation section
    - Link Section
    - Definition Section
    - Global declaration section
    - Function prototype declaration section
    - Main function

User defined function definition section

S.No	Sections	Description
1	Documentation section	We can give comments about the program, creation or modified date, author name etc in this section. The characters or words or anything which are given between "/*" and "*/", won't be considered by Ccompiler for compilation process. These will beignored by C compiler during compilation.Example : /* comment line1 comment line2 comment3 */
2	Link Section	Header files that are required to execute a C program are included in this section
3	Definition Section	In this section, variables are defined and values areset to these variables.
4	Global declaration section	Global variables are defined in this section. When a variable is to be used throughout the program, can be defined in this section.
5	Function Prototype	Function prototype gives many information about a declaration section function like return type, parameter names used
6	Main function	Every C program is started from main function and this function contains two major sections called declaration section and executable section.
7	User defined function section	User can define their own functions in this section which perform particular task as per the user requirement.

Tokens in C

- A C program consists of various tokens and a token is a **keyword, an identifier, a constant, a string literal, or a symbol.**



Identifiers

In C language identifiers are the names given to variables, constants, functions and user-define data. This identifier is defined against a set of rules.

Rules for an Identifier

- An Identifier can only have alphanumeric characters (a-z, A-Z, 0-9) and underscore (_).
- identifier should not start with a number and other special characters (except underscore).

- Identifiers are also case sensitive in C. For example name and Name are two different identifier in C.
- Keywords are not allowed to be used as Identifiers.
- No special characters are permitted, such **(, , : ,. . %, #, !)**
- No space is allowed in between the Identifiers.
- Maximum number of characters allowed for an identifier is 63(31).

Keywords:

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

Character set: In C language characters are grouped into the following categories:

Letters (all alphabets a to z & A to Z).

Digits (all digits 0 to 9).

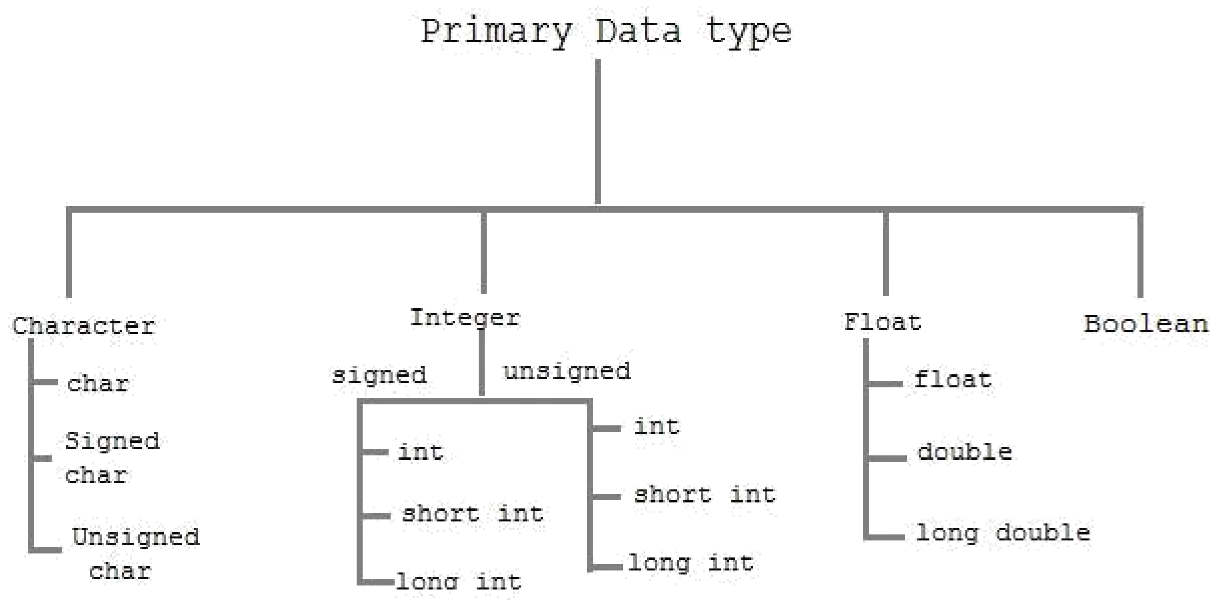
Special characters(such as colon:, semicolon ;, period. , underscore _, ampersand & etc).

DATA TYPES:

Data types in c refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.

The types in C can be classified as follows –

Basic Types: They are arithmetic types and are further classified into: (a) integer types and (b) floating-point types



Enumerated types: They are again arithmetic types and they are used to define variables that can only assign certain discrete integer values throughout the program

The type void: The type specifier void indicates that no value is available.

Derived types : They include (a) Pointer types, (b) Array types, (c) Structure types, (d) Union types and (e) Function types.

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

Void data type in C:

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

Derived data type in C:

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

C – Variable

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

Rules for naming C variable:

- Variable name must begin with letter or underscore.
- Variables are case sensitive
- They can be constructed with digits, letters.
- No special symbols are allowed other than underscore.

  examples : sum, height, _value are some

Declaring & initializing C variable:

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

S.No	**Type**	**Syntax**	**Example**
1	Variable declaration	data_type variable_name;	int x, y, z; char a; float ch;

2	Variable initialization	data_type variable_name =value;	int x = 50, y = 30; char flag = 'x',ch='l';
			float f=45.7;

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

Local variable
Global variable

Local variable in C:

- The scope of local variables will be within the function only.
- These variables are declared within the function and can't be accessed outside the function.

Global variable in C:

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

C – Constants

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

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

Types of C constant:

S.no	**Constant type**	**data type**	**Example**
1	Integer constants	int unsigned int long int long long int	53, 762, -478 etc 5000u, 1000U etc 4,83,647 2,147,483,680
2	Real or Floating point constants	float doule	10.456789 600.1234568
3	Octal constant	int	13 /* starts with 0 */
4	Hexadecimal constant	int	0×90 /* starts with 0x */
5	character constants	char	'A' , 'B', 'C'
6	string constants	char	"ABCD" , "Hai"

1. Example program using const keyword in C:

```
#include <stdio.h>
void main()

 {
 const int height = 100;              /*int constant*/
 const float number = 3.14;
 const char letter = 'A';             /*char constant*/
 const char letter_sequence[10] = "ABC";           /*string constant*/
```

```
  const char backslash_char = '\?';

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

  printf("value of letter_sequence : %s \n", letter_sequence);

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

}
```

C Input and Output Functions

Input : In any programming language input means to feed some data into program. This can be given in the form of file or from command line.

Output : In any programming language output means to display some data on screen, printer or in any file.

- C programming treats all the devices as files. So devices such as the display are addressed in the same way as files and the following three files are automatically opened when a program executes to provide access to the keyboard and screen.

Standard File	**File Pointer**	**Device**
Standard input	stdin	Keyboard
Standard output	stdout	Screen
Standard error	stderr	Your screen

The getchar() and putchar() Functions

- The **int getchar(void)** function reads the next available character from the screen and returns it as an integer. This function reads only single character at a time.

- The **int putchar(int c)** function puts the passed character on the screen and returns the same character. This function puts only single character at a time.

```
#include <stdio.h>
int main( )
{
int c;
printf( "Enter a value :");
c = getchar( );
printf( "\nYou entered: ");
putchar( c );
return 0;
}
```

The gets() and puts() Functions

- The **char *gets(char *s)** function reads a line from stdin into the buffer pointed to by s until either a terminating newline or EOF (End of File).
- The **int puts(const char *s)** function writes the string 's' and 'a' trailing newline to stdout.

```
#include <stdio.h>
int main( )
{
char str[100];
printf( "Enter a value :");
gets( str );
printf( "\nYou entered: ");
puts( str );
return 0;
}
```

The scanf() and printf() Functions

- The **int scanf(const char *format, ...)** function reads the input from the standard input stream stdin and scans that input according to the formatprovided.
- The **int printf(const char *format, ...)** function writes the output to the standard output stream stdout and produces the output according to the format provided.

The format can be a simple constant string, but you can specify ***%s, %d, %c, %f, etc., to print or read strings, integer, character or float respectively.***

```
#include <stdio.h>
main( )
{
char str[100]; int i;
printf( "Enter a value :");
scanf("%s %d", str, &i);
printf( "\nYou entered: %s %d ", str, i);

}
```

Write a Program for Adding two numbers.

Write a Program for converting temperature from degree Celsius to Fahrenheit

3. Write a Program for calculating Kinetic Energy.

<u>C – Operators</u>

- The symbols which are used to perform logical and mathematical operations in a C program are called C operators.
- These C operators join individual constants and variables to form expressions.
- Operators, functions, constants and variables are combined together to form expressions.

Consider the expression **A + B * 5**. Where +, * *are operators*, *A, B are variables* (operands), *5 is constant and A + B * 5 is an expression.*

Types of C operators:

C language offers many types of operators. They are,

- Arithmetic operators
- Relational operators
- Assignment operators
- Logical operators
- Bit wise operators
- Conditional operators (ternary operators)
- Increment/decrement operators
- Special operators

(a). ***Arithmetic operators***

- Arithmetic operators are used to perform arithmetic operations in c programming.
- C Programming Supports 5 Arithmetic Operators.
- Arithmetic Operators are used for "Arithmetic Calculation".

Operator	Meaning	Example
+	Addition Operator	10 + 20 = 30
-	Subtraction Operator	20 – 10 = 10
*	Multiplication Operator	20 * 10 = 200
/	Division Operator	20 / 10 = 2
%	Modulo Operator	20 % 6 = 2

Write a program to understand the floating point arithmetic operations.

Relational operators:

- Relational operators in c programming is used for specifying the relation between two operands such as greater than, less than and equals.
- In C Programming we can compare the value stored between two variables and depending on the result we can follow different blocks using Relational Operator in C.
- Relational operators are used to compare, logical, arithmetic and character expression and each relational operator takes two operands.
- Each operator compares their left side with their right side.It evaluates to 0 if the condition is false and 1 if it is true.

Operator	Meaning	Expression	Interpretation	Value
<	Less than	a<b	FALSE	0
>	Greater than	a>b	TRUE	1
<=	Less than or equal to	a<=b	FALSE	0
>=	Greater than or equal t	a>=b	TRUE	1
==	Equal to	a==b	FALSE	0
!=	Not equal to	a!=b	TRUE	1

c***). Logical Operators:***

- Logical operators are used to combine expressions containing relation operators. To compare more than one condition then we need to use logical operators. In C, there are 3 logical operators:

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

Truth Table

P	Q	P&&Q	P||Q	!P
False	False	False	false	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

d)Assignment Operators:

- An Assignment operator is used to form an assignment expression, which assigns the value to an identifier. The most commonly used assignment operator is = . Assignment expressions that make use of this operator are written in the form

  Identifier=expression;

Operator	Description	Example
=	Simple assignment operator, Assigns values fron right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the resul to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator of the right shift of c by 2)	C <<= 2 is same as C=C<<2 (It assigns the result of the right shift of c by 2)

>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2(It assigns the result of the right shift of c by 2)
&=	Bitwise AND assignment operator	C &= B is same as C = C & B(It assigns the result of the Bitwise AND between B & C)
^=	bitwise exclusive OR and assignment operator	C ^= B is same as C = C ^ B (It assigns the result of the Bitwise Exclusive OR between B & C)
|=	bitwise inclusive OR and assignment operator	C |= B is same as C = C | B (It assigns the result of the Bitwise Inclusive OR between B & C)

(e) Bitwise Operators:

- Bitwise operators are special types of operators that are used in programming the processor. In processor, mathematical operations like: addition, subtraction, addition and division are done using the bitwise operators which makes processing faster and saves power.

Operators	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

1 Write a C program to understand Bitwise operator.

Increment and Decrement operators:

- The **increment (++)** and **decrement (--)** operator are unary operators because they operate on a single operand. the increment operator increment the value of the variable by 1,while decrement operator decrement the value of the variable by 1.

  Ex.
  ```
  ++x   is equivalent to    x=x+1
  --x   is equivalent to    x=x-1
  ```

- These operator should be used only with variables, They can not be used with constant or expressions.

  Ex. The expression ++5 or ++(X+Y+Z) are invalid.

- These operators are of **two types**

1. Prefix-Increment/decrement Operator is written before the operand.
   Ex. ++X or --X
2. Postfix-Increment/decrement Operator is written after the operand.
   Ex. X++ or X--

Prefix-Increment/decrement Operator

- Here first the value of the variable is Incremented or decremented then the new value is used in the operation

- Let us take a variable X=3

  The statement Y=++X; Means first increment the value of X by 1,then assign the value of X to Y. This single statement is equivalent to these two statements

  X= X+1;
  Y=X;
  Now value of X=4 and Y=4

  The statement Y=--X; Means first decrement the value of X by 1,then assign the value of X to Y. This single statement is equivalent to these two statements

  X= X-1;
  Y=X;
  Now value of X=3 and Y=3

1.Write a C program to implement Prefix-Increment/decrement Operator

```
#include<stdio.h>
Int main(void)
{
        Int x=8;
        Printf("x=%d",x);
        Printf("x=%d",++x);
        Printf("x=%d",x);
        Printf("x=%d",--x);
        Printf("x=%d",x);

        Return (0);
}
```

Output:

```
        x=8  x=9  x=9  x=8 x=8
```

Postfix-Increment/decrement Operator
Here first the value of the variable is used in the operation and then value Incremented or decremented is performed.

let us take a variable X=3

The statement Y=X++; Means first the value of X is assigned to Y and then X is incremented. This single statement is equivalent to these two statements

Y=X;
X= X+1;
Now value of X=4 and Y=3

The statement Y=X--; Means first the value of X is assigned to Y and then X is incremented. This single statement is equivalent to these two statements

Y=X;
X= X-1;
Now value of X=3 and Y=4

1.Write a C program to implement Postfix-Increment/decrement Operator

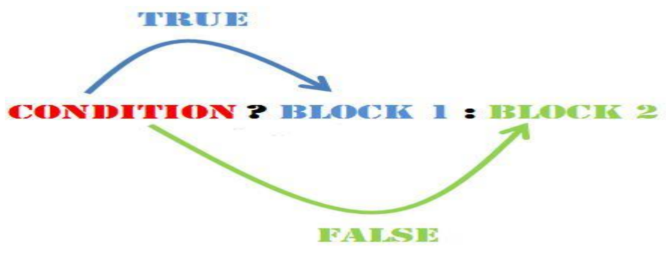
(g). Conditional Operator (?:):

The conditional expression can be used as shorthand for some if-else statements. It is a ternary operator. This operator consists of two symbols: the

question mark (?) and the colon (:).

The general syntax of the conditional operator is:

Identifier = (test expression)? Expression1: Expression2;



This is an expression, not a statement, so it represents a value. The operator works by evaluating test expression (condition). If it is true (non-zero), it evaluates and returns expression1 (Block1). Otherwise, it evaluates and returns expression2 (Block2).

1.Write a program for Conditional operator.

(i) Special Operators:

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

EXPRESSIONS:

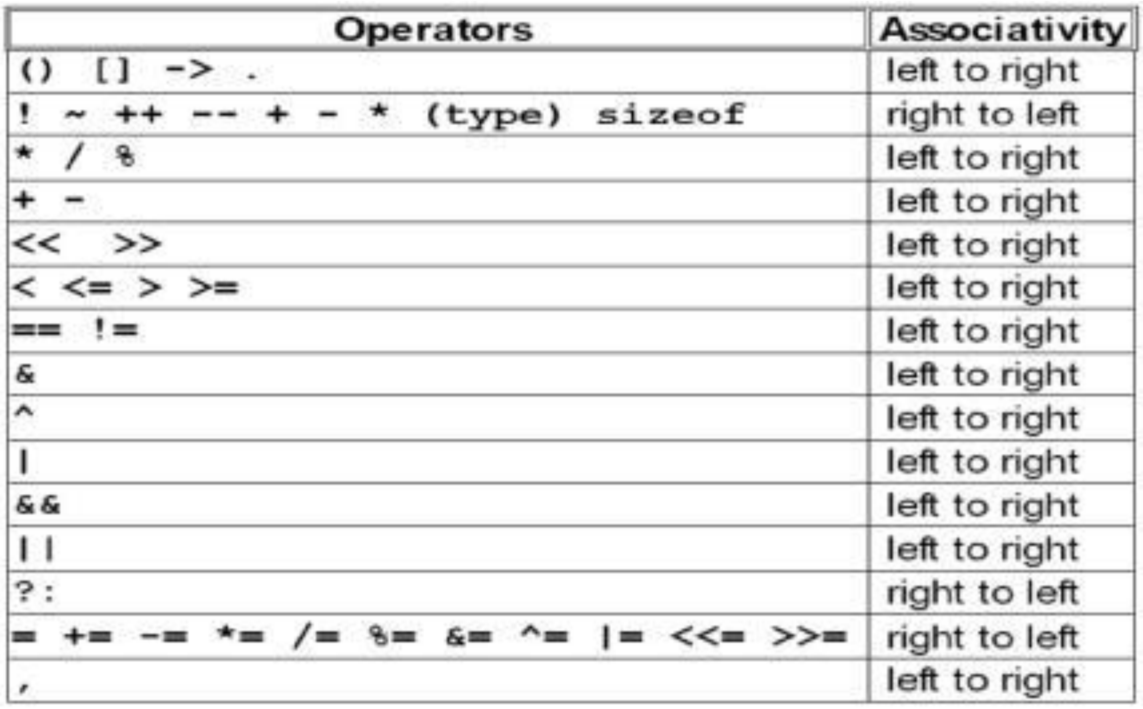
- An expression is a sequence of operands and operators that reduces to a single value.
- Expression can be simple or complex.
- An operator is a syntactical token that requires an action be taken.
- An operand is an object on which an operation is performed.

A **simple expression** contains only one operator. E.g: 2 + 3 is a simple expression whose value is 5.A complex expression contains more that one operator. E.g: 2 + 3 * 2.

To evaluate a **complex expression** we reduce it to a series of simple expressions. In this first we will evaluate the simple expression 3 * 2 (6)and then the expression 2 + 6,giving a result of 8.

The order in which the operators in a complex expression are evaluated is determined by a set of priorities known as **precedence**, the higher the precedence ,the earlier the expression containing the operator is evaluated.

The following table shows the precedence and Associativity of operators:



Operators	Associativity
() [] -> .	left to right
! ~ ++ -- + - * (type) sizeof	right to left
* / %	left to right
+ -	left to right
<< >>	left to right
< <= > >=	left to right
== !=	left to right
&	left to right
^	left to right
|	left to right
&&	left to right
||	left to right
?:	right to left
= += -= *= /= %= &= ^= |= <<= >>=	right to left
,	left to right

TYPE CONVERSION:

In an expression that involves two different data types , such as multiplying an integer and a floating point number to perform these evaluations ,one of the types must be converted.

We have two types of conversions:

1.Implicit Type Conversion 2.Explicit Type Conversion

IMPLICIT TYPE CONVERSION: When the types of the two operands in a binary expression are different automatically converts one type to another .This is known as implicit type conversion.

EXPLICIT TYPE CONVERSION: Explicit type conversion uses the unary cast operator ,to convert data from one type to another. To cast data from one type to another ,we specify the new type in parentheses before the value we want converted.

For example ,to convert an integer ,a , to a float, we code the expression **like (float) a**

1. **write a program to check implicit and explicit type conversion**

DECICSION MAKING OR CONTROL STATEMENTS:

IF AND SWITCH STATEMENTS:

We have a number of situations where we may have to change the order of execution of statements based on certain conditions or repeat a group of statements until certain specified conditions are met.

The **if statement is a two way decision statement** and is used in conjunction with an expression. It takes the following form
If the test expression is true then the statement block after if is executed otherwise it is not executed

Simple if:

```
if (test expression)

{

statement block;

}

statement–x ;
```

only statement–x is executed.

The if –else statement:

If your have another set of statement to be executed if condition is false then if-else is used

```
if (test expression)

{
statement block1;

}

else

{

statement block2;

}

statement –x ;
```

1.Write Algorithm, Program and draw a flowchat for Roots of quadratic equation.

Nesting of if..else statement :

If more than one if else statement with in if else

```
if(text cond1)

{
        if (test expression2

        {
        statement block1;
        }
        else
        {
        statement block 2;
        }
}
else
{
        statement block2;
}
statement-x ;
```

if else ladder:

- The nesting of if-else depends upon the conditions with which we have to deal.
- This statement is similar to switch statement.

```
if(condition1)
        statement1;
else if(condition2)
        statement 2;
else if(condition3)
        statement n;
else
        default statement.
statement-x;
```

THE SWITCH STATEMENT:

- switch statement used when user have to choose from more than two option/choices (means multiple option)

```
switch(expression)

{.
case value-1:
        block-1
        break;
case value-2:
        block-2
        break;
--------
--------
default:
        default block;
        break;
}
statement–x;
```

1.Write a Program for Arithmetic operations using switch case.

LOOPING or ITERATION or REPETETION STATEMENTS:

Some times we require a set of statements to be executed repeatedly until a condition is met.

We have two types of looping structures. One in which condition is tested

before entering the statement block called entry control. The other in which condition is checked at exit called exit controlled loop.

First initialization is done once, then condition is evaluated and if it is true then body of loop is executed. After execution of body the control goes to

increment/ decrement part then condition is once again evaluated and if true body is executed once again. This goes on until test condition becomes false.

WHILE STATEMENT:

```
while(test condition)

{

body of the loop

}
```

1. **write a program which describe while loop**

DO WHILE STATEMENT:

The while loop does not allow body to be executed if test condition is false. The do while is an exit controlled loop and its body is executed at least once.

```
do

{

body

}while(test condition);
```

write a program which describe do- while loop

THE FOR LOOP:

It is also an entry control loop that provides a more concise structure

```
for(initialization; test control; increment/decrement)
{
body of loop

}
```

1 Write a program which describe for loop

1. **Write a program for printing a multiplication table for given number using all loops(while, do-while, for loop)**

OTHER STATEMENTS:

BREAK STATEMENT:

- This is a simple statement. It only makes sense if it occurs in the body of a switch, do, while or for.
- When it is executed the control of flow jumps to the statement immediately following the body of the statement containing the break.
- Its use is widespread in switch statements, where it is more or less essential to get the control.

CONTINUE STATEMENT:

- This statement meaning is “Skip the following statements and do the next iteration”.
- Executing a continue starts the next iteration of the loops like do, while or for statement immediately.
- The use of continue is largely restricted to the top of loops, where a decision has to be made whether or not to execute the rest of the body of the loop.

Example it ensures that division by zero (which gives undefined behavior) doesn't happen.

```
#include <stdio.h>
#include <stdlib.h>
main(){

int i;
for(i = -10; i < 10; i++)
{
        if(i == 0)
            Continue;
        printf("%f\n", 15.0/i);

}
```

GOTO STATEMENT :

- In C, it is used to escape from multiple nested loops
- You will need a label when you use goto statement.

```
goto L1;
/* whatever you like here */
L1: /* anything else */
```

```
#include <stdio.h>
int main()
{
   int sum=0,;
   for(int i = 0; i<=10; i++){
            sum = sum+i;
            if(i==5){
               goto addition;
            }
   }
```

```
 addition:
   printf("%d", sum);
}
```

Output:

C Pointers: stdin, stdout, stderr

Linux is built being able to run instructions from the command line using switches to create the output.One of the ways to make use of this is by using the three special file descriptors - stdin, stdout and stderr

stdin

Generally standard input, referred to as "stdin", comes from the keyboard.When you type stuff or data, you're typing it on stdin (a standard input terminal). A standard input device, which is usually the keyboard, but Linux also allows you take standard input from a file.

stdout

Standard output, as created at process creating time, goes to the console, your terminal or an X terminal. Exactly where output is sent clearly depends on where the process originated.Our console or terminal should be the device that is accepting the output. Running the command:

stderr

The final component in this dialog of file descriptors is standard error.Every command could send it's output to one of two places: a) it could be valid output or b) it could be an error message.It does the same with the errors as it does with the standard output; it sends them directly to your terminal screen.

Command Line arguments

- The most important function of C is main() function.
- It is mostly defined with a return type of int and without parameters :int main() { /* ... */ }
- We can also give command-line arguments in C .
- Command-line arguments are given after the name of the program in command-line shell of Operating Systems.
- To pass command line arguments, we typically define main() with two arguments : first argument is the number of command line arguments and second is list of command-line arguments.

```
int main(int argc, char *argv[]) { /* ... */ }

                  or

int main(int argc, char **argv) { /* ... */ }
```

- argc (Argument Count) is int and stores number of command-line arguments passed by the user including the name of the program. So if we pass a value to a program, value of argc would be 2 (one for argument and one for program name)

- The value of argc should be non negative.

- argv(Argument Vector) is array of character pointers listing all the arguments.

- Argv[0] is the name of the program , After that till argv[argc-1] every element is command -line arguments.

```
#include <stdio.h>

int main(int argc, char** argv)
{
   Printf("You have entered  %d \n" ,argc);
   Printf(" arguments:\n");

   for (int i = 0; i < argc; ++i)
       printf("%s\n", argv[i]);

   return 0;
}
```

Output:

<u>Storage Classes</u>

- Storage classes of c will provide following information to compiler.
- storage area of variable.
- Scope of variable i.e in which block the variable is visible.
- Lifetime of the variable i.e how long the variable will be there in active mode.
- Default value of the variable if it is not initialized.

Depending on the behaviour and storage area storage classes are classified into 2 types.

1. Automatic Storage class
2. Static storage class

Automatic Storage class:

- This storage class variables are created automatically and destroyed automatically.
- Automatic storage class variables will be stored in stack area of data segment or CPU register.

- Under automatic storage class we are having 2 types of storage class specifies
    1. Auto
    2. Register

Static storage class:

- Static storage class variable are created only ones and throughout the program it will be there in active mode only.
- Static storage class variables are stored in static area of data segment.
- Under static storage class we have 2 types of storage class specifiers
    1. Satic
    2. Extern

Code Properties:

Type	**Scope**	**Life**	**Default Value**
Auto	Block	Block	Garbage Value
Static	Block	Program	0
Extern	Program	Program or All functions	0
Register	Block	Block	Garbage Value

Note:
By default any kind of variable storage class specifier is auto.
In C- Programming language there are 4 – types of scope are available i.e. body, function, program and file scope.

```
Int a;
Short int a;
Signed short int a;
Auto        signed      short       int         a;
|             |           |           |           |
|             |           |           |           |
Storage     Sign        Size        Type        Variable
```

Register Variables:
It is a special kind of variable which stores in CPU register.
The advantage of register variables are faster than remaining variables.

Limitations:
When we are working with register variable we can't create 'n' number of register variables. i.e., depending upon CPU capacity 4.6 are maximum variables.
In TC.3.0 we can't access the address of register variables.
Pointer or pointer related concepts are not applied to register variables.

Examples:

```
Void main()
{
Register int a=10;
++a;
Printf("\n value of a:%d",a);
Printf("Enter a value:");
Scanf("%d",&a);
--a;
Printf("value of a:%d",a);
Getch();
}
```

Output: Error

1.write a Program for storage Classes

Number System:

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

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

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

Decimal Number System

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

Example: the decimal number 1234 consists of the digit 4 in the units position, 3 in the tens position, 2 in the hundreds position, and 1 in the thousands position. Its value can be written as

- (1 x 1000)+ (2 x 100)+ (3 x 10)+ (4 x l)
- (1 x 10^3)+ (2 x 10^2)+ (3 x 10^1)+ (4 x $l0^0$)
- 1000 + 200 + 30 + 4
- 1234

Other Number System:

1 **Binary Number System**: Base 2. Digits used : 0, 1

2 **Octal Number System**: Base 8. Digits used : 0 to 7

3 **Hexa Decimal Number System**: Base 16. Digits used: 0 to 9, Letters used : A- F