PROGRAMMING FOR PROBLEM SOLVING USING C
UNIT I

Introduction to Computers: Computer Systems, Computing Environments, Computer languages, Creating and running Programs, Computer Numbering System, Storing Integers, Storing Real Numbers

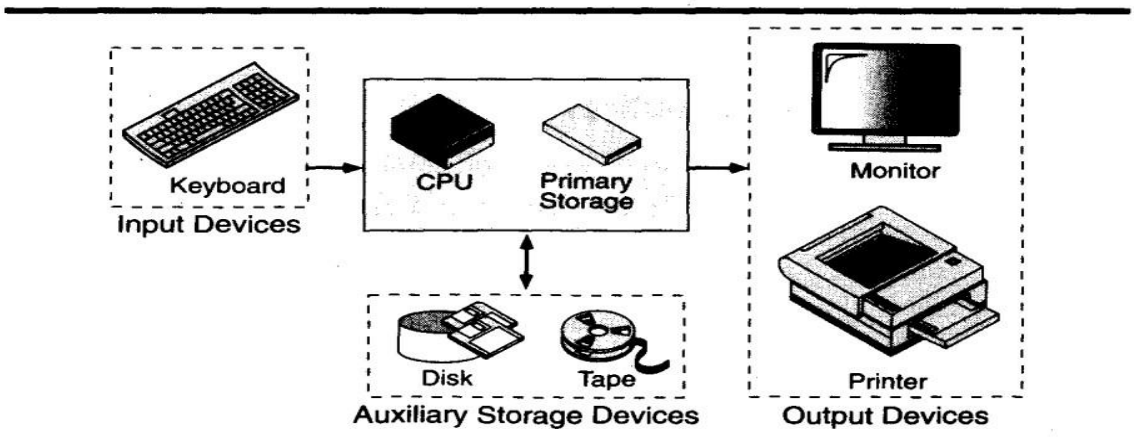
Introduction to the C Language: Background, C Programs, Identifiers, Types, Variable, Constants, Input/output, Programming Examples, Scope, Storage Classes and Type Qualifiers, Tips and Common Programming Errors Key Terms, Summary, Practice Seat.

Structure of a C Program: Expressions Precedence and Associativity, Side Effects, Evaluating Expressions, Type Conversion Statements, Simple Programs, Command Line Arguments Tips and Common Errors, Key Terms, Summary, Practice Sets.

➢ Computer Systems:-

A computer is a system made of two major components: hardware and software. The computer hardware is the physical equipment. The software is the collection of programs (instructions) that allow the hardware to do its job.

❖ **Computer Hardware: -** The hardware component of the computer system consists of five parts: input devices, central processing unit (CPU) ,primary storage, output devices, and auxiliary storage devices.



Basic Hardware Components

- The **input device** is usually a keyboard where programs and data are entered into the computers. Examples of other input devices include a mouse, a pen or stylus, a touch screen, or an audio input unit.
- The **central processing unit (CPU**) is responsible for executing instructions such as arithmetic calculations, comparisons among data, and movement of data inside the system.
- The **output device** is usually a monitor or a printer to show output. If the output is shown on the monitor, we say we have a **soft copy**. If it is printed on the printer, we say we have a hard copy.
- **Auxiliary storage**, also known as **secondary storage**, is used for both input and output. It is the place where the programs and data are stored permanently. When we turn off the computer, or programs and data remain in the secondary storage, ready for the next time we need them.

❖ **Computer Software :-**

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

- System software manages the computer resources .It provides the interface between the hardware and the users.
- Application software, on the other hand is directly responsible for helping users solve their problems.

➢ Computing Environments:-

Computing Environment is a collection of computers / machines, software, and networks that support the processing and exchange of electronic information meant to support various types of computing solutions. With the advent if technology the computing environments have been improved.

Types of Computing Environments:-

1. Personal Computing Environment
2. Time sharing Environment
3. Client Server Computing Environment
4. Distributed Computing

1. Personal Computing Environment:-

Personal means, all the computer stuff will be tied together i.e computer is completely ours, no other Connections.

2. Time sharing Environment:-

In computing, time-sharing is the sharing of a computing resource among many users by means of multi programming and multi-tasking at the same time. Mam users are connected to one or more computers.

3. **Client Server Computing Environment**:-

A client/server system is "a networked computing model that distributes processes between clients and servers, which supply the requested service." A client/server network connects many computers, called clients, to a main computer, called a server. Whenever client requests for something, server receives the request and process it.

4. **Distributed Computing**:-

A Distributed Computing Environment Provides a seamless integration of computing functions between different servers and clients. the servers are connected by internet all over the world.

➢ 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.

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.

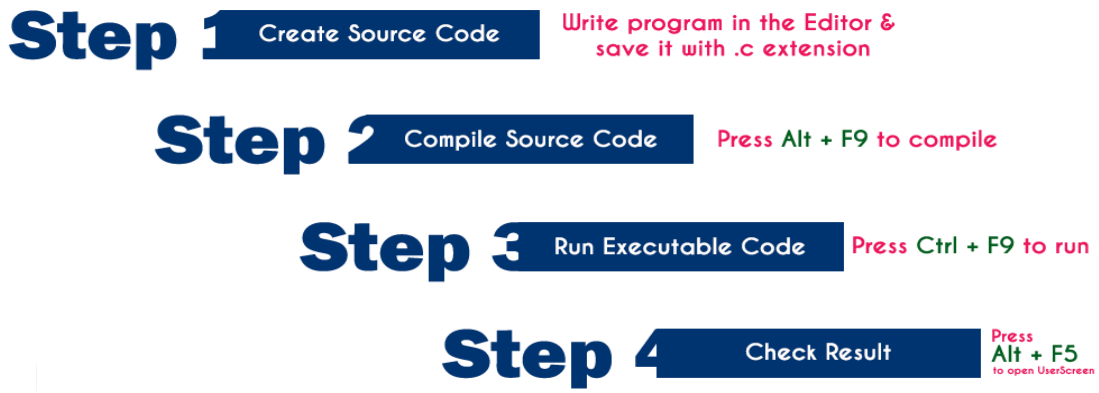
❖ **High Level Languages:-**

Symbolic languages greatly improved programming effificiency; they still required programmers to concentrate on the hardware that they were using.

Working with symbolic languages was also very tedious because each machine instruction has to be individually coded. The desire to improve programmer efficiency and to change the focus from the computer to the problem being solved led to the development of high-level language.

➢ Creating and running Programs:-

- Generally, the programs created using programming languages like C, C++, Java, etc., are written using a high-level language like English. But, the computer cannot understand the high-level language.
- It can understand only low-level language. So, the program written in high-level language needs to be converted into the low-level language to make it understandable for the computer. This conversion is performed using either Interpreter or Compiler.
- Popular programming languages like C, C++, Java, etc., use the compiler to convert high-level language instructions into low-level language instructions.
- To create and execute C programs in Windows Operating System, we need to install Turbo C software. We use the following steps to create and execute C programs in Windows **OS...**



Step 1: Creating Source Code

Source code is a file with C programming instructions in high level language. To create source code, we use any text editor to write the program instructions. The instructions written in the source code must follow the C programming language rules. The following steps **are used to create source code file in Windows OS...**

- Click on **Start** button
- Select **Run**
- Type **cmd** and press Enter
- Type **cd c:\TC\bin** in the command prompt and press **Enter**
- Type **TC** press **Enter**
- Click on **File -> New** in C Editorwindow
- Type the **program**
- Save it as **FileName.c** (Use shortcut key **F2** to save)

Step 2: Compile Source Code (Alt + F9)

Compilation is the process of converting high level language instructions into low level language instructions. We use the shortcut key **Alt + F9** to compile a C program in **Turbo C**.

Whenever we press **Alt + F9**, the source file is going to be submitted to the Compiler. On receiving a source file, the compiler first checks for the Errors. If there are any Errors then compiler returns List of Errors, if there are no errors then the source code is converted into **object code** and stores it as file with **.obj** extension. Then the object code is given to the **Linker**. The Linker combines

both the **object code** and specified **header file** code and generates an **Executable file** with **.exe** extension.

Step 3: Executing / Running Executable File (Ctrl + F9)

After completing compilation successfully, an executable file is created with **.exe** extension. The processor can understand this **.exe** file content so that it can perform the task specified in the source file.

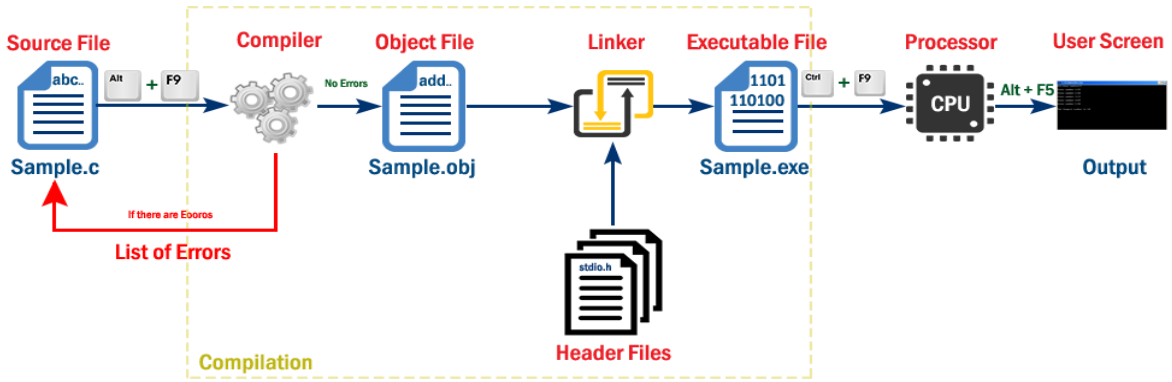
We use a shortcut key **Ctrl + F9** to run a C program. Whenever we press **Ctrl + F9**, the **.exe** file is submitted to the **CPU**. On receiving **.exe** file, **CPU** performs the task according to the instruction written in the file. The result generated from the execution is placed in a window called **User Screen**.

Step 4: Check Result (Alt + F5)

After running the program, the result is placed into **User Screen**. Just we need to open the User Screen to check the result of the program execution. We use the shortcut key **Alt + F5** to open the User Screen and check the result.

Execution Process of a C Program

When we execute a C program it undergoes with following process...



- The file which contains c program instructions in high level language is said to be source code. Every c program source file is saved with .c extension, for example Sample.c.

- Whenever we press Alt + F9 the source file is submitted to the compiler. Compiler checks for the errors, if there are any errors, it returns list of errors, otherwise generates object code in a file with name Sample.obj and submit it to the linker.
- Linker combines the code from specified header file into object file and generates executable file as Sample.exe. With this compilation process completes.
- Now, we need to Run the executable file (Sample.exe). To run a program we press Ctrl + F9. When we press Ctrl + F9 the executable file is submitted to the CPU.
- Then CPU performs the task according to the instructions written in that program and place the result into UserScreen.
- Then we press Alt + F5 to open UserScreen and check the result of the program.

Overall Process:-

- Type the program in C editor and save with **.c extension** (Press **F2** to save).
- Press **Alt + F9** to compile the program.
- If there are errors, correct the errors and recompile the program.
- If there are no errors, then press **Ctrl + F9** to execute / run the program.
- Press **Alt + F5** to open User Screen and check the result.

➢ Computer Numbering System:-

The technique to represent and work with numbers is called **number system**. **Decimal number system** is the most common number system. Other popular number systems include **binary number system, octal number system, hexadecimal number system,** etc.

Decimal Number System:-

Decimal number system is a **base 10** number system having 10 digits from 0 to 9. This means that any numerical quantity can be represented using these 10 digits. Decimal number system is also a **positional value system**.

Binary Number System:-

The easiest way to vary instructions through electric signals is two-state system – on and off. On is represented as 1 and off as 0, though 0 is not actually no signal but signal at a lower voltage. The number system having just these two digits – 0 and 1 – is called **binary number system**.

Octal Number System:-

Octal number system has eight digits – 0, 1, 2, 3, 4, 5, 6 and 7. Octal number system is also a positional value system with where each digit has its value expressed in powers of 8, as shown here −

8^5	8^4	8^3	8^2	8^1	8^0

Decimal equivalent of any octal number is sum of product of each digit with its positional value.

$726_8 = 7\times8^2 + 2\times8^1 + 6\times8^0$

$= 448 + 16 + 6$

$= 470_{10}$

Hexadecimal Number System:-

Octal number system has 16 symbols – 0 to 9 and A to F where A is equal to 10, B is equal to 11 and so on till F. Hexadecimal number system is also a positional value system with where each digit has its value expressed in powers of 16, as shown here −

16^5	16^4	16^3	16^2	16^1	16^0

➢ Storing Integers:-

Integers are commonly stored using a word of memory, which is 4 bytes or 32 bits, so integers from 0 up to 4,294,967,295 ($2^{32} - 1$) can be stored.

Below are the integers 1 to 5 stored as four-byte values (each row represents one integer).

```
0  :  00000001 00000000 00000000 00000000  |  1
 4  :  00000010 00000000 00000000 00000000  |  2
 8  :  00000011 00000000 00000000 00000000  |  3
12  :  00000100 00000000 00000000 00000000  |  4
16  :  00000101 00000000 00000000 00000000  |  5
```

This may look a little strange; within each byte (each block of eight bits), the bits are written from right to left like we are used to in normal decimal notation, but the bytes themselves are written left to right! It turns out that the computer does not mind which order the bytes are used (as long as we tell the computer what the order is) and most software uses this left to right order for bytes.[7.3]

Two problems should immediately be apparent: this does not allow for negative values, and very large integers, 2^{32} or greater, cannot be stored in a word of memory.

➢ Real numbers:-

Real numbers (and rationals) are much harder to store digitally than integers.

Recall that k bits can represent 2^k different states. For integers, the first state can represent 0, the second state can represent 1, the third state can represent 2, and so on. We can only go as high as the integer 2^k - 1, but at least we know that we can account for all of the integers up to that point.

Unfortunately, we cannot do the same thing for reals. We could say that the first state represents 0, but what does the second state represent? 0.1? 0.01? 0.00000001? Suppose we chose 0.01, so the first state represents 0, the second state represents 0.01, the third state represents 0.02, and so on. We can now only go as high as 0.01 x (2^k - 1), *and* we have missed all of the numbers between 0.01 and 0.02 (and all of the numbers between 0.02 and 0.03, and infinitely many others).

➢ 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 it's 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. In other words, the majority of the code that makes up UNIX does not know and does not care which computer it is actually running on.
- C was first designed by Dennis Ritchie for use with UNIX on DEC PDP-11 computers. The language evolved from Martin Richard's BCPL, and one of its earlier forms was the B language, which was written by Ken Thompson for the DEC PDP-7. The first book on C was *The C Programming Language* by Brian Kernighan and Dennis Ritchie, published in 1978.
- In 1983, the American National Standards Institute (ANSI) established a committee to standardize the definition of C. The resulting standard is known as *ANSI C,* and it is the recognized standard for the language, grammar, and a core set of libraries. The syntax is slightly different from the original C language, which is frequently called K&R for Kernighan and Ritchie. There is also an ISO (International Standards Organization) standard that is very similar to the ANSI standard.
- 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."

➢ BASIC STRUCTURE OF C LANGUAGE:-

- The program written in C language follows this basic structure. The sequence of sections should be as they are in the basic structure. A C program should have one or more sections but the sequence of sections is to be followed.

1. Documentation section
2. Linking section
3. Definition section
4. Global declaration section
5. Main function section

```
{
Declaration section
Executable section
}
```

6.Sub program or function section

1. DOCUMENTATION SECTION : comes first and is used to document the use of logic or reasons in your program. It can be used to write the program's objective, developer and logic details. The documentation is done in C language with /* and */ . Whatever is written between these two are called comments.

2. LINKING SECTION : This section tells the compiler to link the certain occurrences of keywords or functions in your program to the header files specified in this section.
e.g. #include <stdio.h>

3. DEFINITION SECTION : It is used to declare some constants and assign them some value.
e.g. #define MAX 25
Here #define is a compiler directive which tells the compiler whenever MAX is found in the program replace it with 25.

4. GLOBAL DECLARATION SECTION : Here the variables which are used through out the program (including main and other functions) are declared so as to make them global(i.e accessible to all parts of program)
e.g. int i; (before main())

5. MAIN FUNCTION SECTION : It tells the compiler where to start the execution from

```
main()
{
   point from execution starts
}
```

main function has two sections

1. declaration section : In this the variables and their data types are declared.
2. Executable section : This has the part of program which actually performs the task we need.

6. SUB PROGRAM OR FUNCTION SECTION : This has all the sub programs or the functions which our program needs.

SIMPLE 'C' PROGRAM:

```
/* simple program in c */
#include<stdio.h>
main()
{
printf("welcome to c programming");
} /* End of main */
```

IDENTIFIERS :-

- Names of the variables and other program elements such as functions, array, etc, are known as identifiers.

There are few rules that govern the way variable are named (identifiers).

1. Identifiers can be named from the combination of A-Z, a-z, 0-9, _(Underscore).
2. The first alphabet of the identifier should be either an alphabet or an underscore. digit are not allowed.
3. It should not be a keyword.
Eg: name, ptr, sum
After naming a variable we need to declare it to compiler of what data type it is .
The format of declaring a variable is

Data-type id1, id2, ...idn;

where data type could be float, int, char or any of the data types.
id1, id2, id3 are the names of variable we use. In case of single variable no commas are required.

```
Eg      float a, b, c;
        int e, f, grand total;
        char present_or_absent;
```

DATA TYPES :-

To represent different types of data in C program we need different data types. A data type is essential to identify the storage representation and the type of operations that can be performed on that data. C supports four different classes of data types namely

1. Basic Data types
2. Derives data types
3. User defined data types
4. Pointer data types

BASIC DATA TYPES:

All arithmetic operations such as Addition , subtraction etc are possible on basic data types.

```
E.g.: int a,b;
      Char c;
```

DERIVED DATA TYPES:-

Derived datatypes are used in 'C' to store a set of data values. Arrays and Structures are examples for derived data types.

```
Ex:   int a[10];
      Char name[20];
```

USER DEFINED DATATYPES:

C Provides a facility called typedef for creating new data type names defined by the user. For Example ,the declaration ,

typedef int Integer;

makes the name Integer a synonym of int.Now the type Integer can be used in declarations ,casts,etc,like,

Integer num1,num2;

Which will be treated by the C compiler as the declaration of num1,num2as int variables.

"typedef" ia more useful with structures and pointers.

POINTER DATA TYPES:-

Pointer data type is necessary to store the address of a variable.

➢ VARIABLES :-

A quantity that can vary during the execution of a program is known as a variable. To identify a quantity we name the variable for example if we are calculating a sum of two numbers we will name the variable that will hold the value of sum of two numbers as 'sum'.

➢ CONSTANTS : -

A quantity that does not vary during the execution of a program is known as a constant supports two types of constants namely Numeric constants and character constants.

NUMERIC CONSTANTS:

1. Example for an integer constant is 786,-127
2. Long constant is written with a terminal 'l'or 'L',for example 1234567899L is a Long constant.

CHARACTER CONSTANTS:-

A character constant is written as one character with in single quotes such as 'a'. The value of a character constant is the numerical value of the character in the machines character set.

➢ INPUT AND OUTPUT STATEMENTS :-

The simplest of input operator is getchar to read a single character from the input device.

varname=getchar();

you need to declare varname.

The simplest of output operator is putchar to output a single character on the output device.

putchar(varname)

The getchar() is used only for one input and is not formatted. Formatted input refers to an input data that has been arranged in a particular format, for that we have scanf.

scanf("control string", arg1, arg2,...argn);

Control string specifies field format in which data is to be entered.

arg1, arg2... argn specifies address of location or variable where data

Eg scanf("%d%d",&a,&b);

%d used for integers
%f floats
%l long
%c character

for formatted output you use printf

printf("control string", arg1, arg2,...argn);

/* program to exhibit i/o */

```
#include<stdio.h>
main()
{
int a,b;
float c;
printf("Enter any number");
a=getchar();
printf("the char is ");
putchar(a);
printf("Exhibiting the use of scanf");
printf("Enter three numbers");
scanf("%d%d%f",&a,&b,&c);
printf("%d%d%f",a,b,c);
}
```

- **Scope:-**

  A scope in any programming is a region of the program where a defined variable can have its existence and beyond that variable it cannot be accessed. There are three places where variables can be declared in C programming language –

  - Inside a function or a block which is called **local** variables.

- Outside of all functions which is called **global** variables.
- In the definition of function parameters which are called **formal** parameters.

Local Variables:-

Variables that are declared inside a function or block are called local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own.

Global Variables:-

Global variables are defined outside a function, usually on top of the program. Global variables hold their values throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program.

A global variable can be accessed by any function.

➢ Storage Classes:-

A storage class defines the scope (visibility) and life-time of variables and/or functions within a C Program. They precede the type that they modify. We have four different storage classes in a C program −

- auto
- register
- static
- extern

The auto Storage Class

The **auto** storage class is the default storage class for all local variables.

```
{
   int  mount;
   auto int month;
}
```

The example above defines two variables with in the same storage class. 'auto' can only be used within functions, i.e., local variables.

The register Storage Class:-

The **register** storage class is used to define local variables that should be stored in a register instead of RAM. This means that the variable has a maximum size equal to the register size (usually one word) and can't have the unary '&' operator applied to it (as it does not have a memory location).

```
{
   register int  miles;
}
```

The register should only be used for variables that require quick access such as counters.

The static Storage Class:-

The **static** storage class instructs the compiler to keep a local variable in existence during the life-time of the program instead of creating and destroying it each time it comes into and goes out of scope..

The extern Storage Class

The **extern** storage class is used to give a reference of a global variable that is visible to ALL the program files. When you use 'extern', the variable cannot be initialized however, it points the variable name at a storage location that has been previously defined.

Type Qualifiers:-

The keywords which are used to modify the properties of a variable are called type qualifiers.

TYPES OF C TYPE QUALIFIERS:
There are two types of qualifiers available in C language. They are,

1. const
2. volatile

1. CONST KEYWORD:

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

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

VOLATILE KEYWORD:

- When a variable is defined as volatile, the program may not change the value of the variable explicitly.
- But, these variable values might keep on changing without any explicit assignment by the program. These types of qualifiers are called volatile.

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

Tips and Common Programming Errors:-

When you start writing your code in C, C++ or any other programming language, your first objective might be to write a program that works.

After you accomplished that, the following are few things you should consider to enhance your program.

1. Security of the program
2. Memory consumption
3. Speed of the program (Performance Improvement)

This article will give some high-level ideas on how to improve the speed of your program.

Few general points to keep in mind:-

- You could optimize your code for performance using all possible techniques, but this might generate a bigger file with bigger memory footprint.
- You might have two different optimization goals, that might sometimes conflict with each other. For example, to optimize the code for performance might conflict with optimize the code for less memory footprint and size. You might have to find a balance.
- Performance optimization is a never-ending process. Your code might never be fully optimized. There is always more room for improvement to make your code run faster.
- Sometime we can use certain programming tricks to make a code run faster at the expense of not following best practices such as coding standards, etc. Try to avoid implementing cheap tricks to make your code run faster.

➢ Expressions:-

An expression is a combination of variables constants and operators written according to the syntax of C language.

In C every expression evaluates to a value i.e., every expression results in some value of a certain type that can be assigned to a variable.

Evaluation of Expressions:-

Expressions are evaluated using an assignment statement of the form.

```
Variable = expression;
```

Variable is any valid C variable name. When the statement is encountered, the expression is evaluated first and then replaces the previous value of the variable on the left hand side.

. All variables used in the expression must be assigned values before evaluation is attempted.

Example of evaluation statements are

X=a*b-c
Y=b/c*a
Z=a-b/c+d;

➢ Precedence and Associativity:-

Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator.

For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has a higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right

Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Side Effects of c:-

In C and more generally in computer science, a function or expression is said to have a **side effect** if it modifies a state outside its scope or has an *observable* interaction with its calling functions or the outside world. By convention, returning a value has an effect on the calling function, but this is usually not considered as a side effect.

Some side effects are:

- Modification of a global variable or static variable
- Modification of function arguments
- Writing data to a display or file
- Reading data
- Calling other side-effecting functions

Type Conversion Statements:-

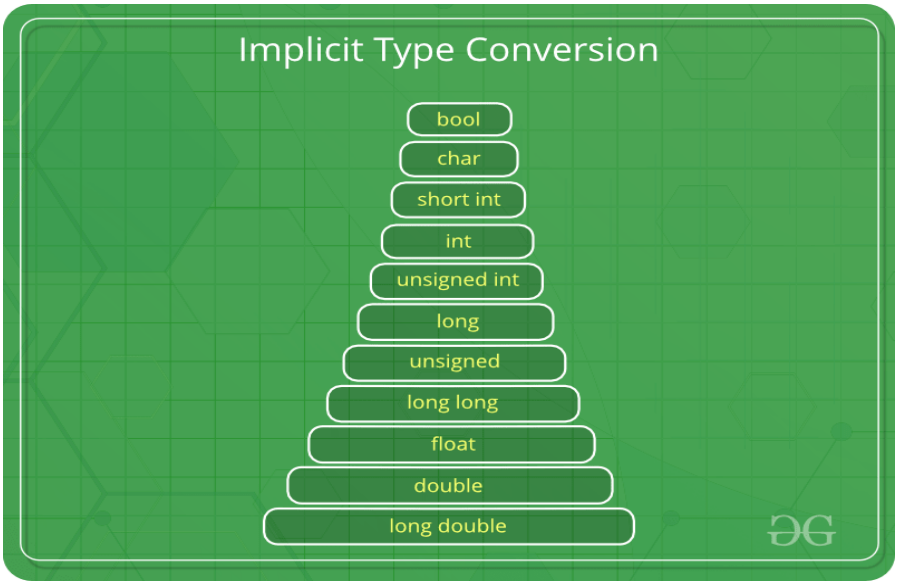
Typecasting is converting one data type into another one. It is also called as data conversion or type conversion. It is one of the important concepts introduced in 'C' programming.

'C' programming provides two types of type casting operations:

1. Implicit type casting
2. Explicit type casting

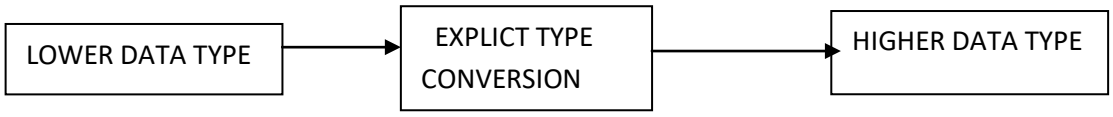
Implicit type casting

Implicit type casting means conversion of data types without losing its original meaning. This type of typecasting is essential when you want to change data types **without** changing the significance of the values stored inside the variable.



Explicit type casting

In implicit type conversion, the data type is converted automatically. There are some scenarios in which we may have to force type conversion. Suppose we have a variable div that stores the division of two operands which are declared as an int data type.



SIMPLE PROGRAM:-

1.Prime Number program in C

```
1. #include<stdio.h>
2. int main(){
3. int n,i,m=0,flag=0;
4. printf("Enter the number to check prime:");
5. scanf("%d",&n);
6. m=n/2;
7. for(i=2;i<=m;i++)
8. {
9. if(n%i==0)
10.{
11.printf("Number is not prime");
12.flag=1;
13.break;
14.}
15.}
16.if(flag==0)
17.printf("Number is prime");
18.return 0;
19. }
```

Output:

```
Enter the number to check prime:56
Number is not prime
Enter the number to check prime:23
Number is prime
```

- **Command Line Arguments:-**

It is possible to pass some values from the command line to your C programs when they are executed.

These values are called command line arguments and many times they are important for your program especially when you want to control your program from outside instead of hard coding those values inside the code.

The command line arguments are handled using main() function arguments where argc refers to the number of arguments passed, and argv[] is a pointer array which points to each argument passed to the program.

PROGRAMMING FOR PROBLEM SOLVING USING C
UNIT II

Bitwise Operators: Exact Size Integer Types, Logical Bitwise Operators, Shift Operators, Tips and Common Programming Errors, Key Terms, Summary, Practice Set.

Selection & Making Decisions: Logical Data and Operators, Two Way Selection, Multiway Selection, More Standard Functions, Tips and Common Programming Errors, Key Terms, Summary, Practice Set.

Repetition: Concept of Loop, Pretest and Post-test Loops, Initialization and Updating, Event and Counter Controlled Loops, Loops in C, Other Statements Related to Looping, Looping Applications, Programming Example The Calculator Program, Tips and Common Programming Errors, Key Terms, Summary, Practice Set.

➢ Bitwise Operators:-

- These operators are used to perform bit operations. Decimal values are converted into binary values which are the sequence of bits and bit wise operators work on these bits.
- Bit wise operators in C language are & (bitwise AND), | (bitwise OR), ~ (bitwise NOT), ^ (XOR), << (left shift) and >> (right shift).

 TRUTH TABLE FOR BIT WISE OPERATION & BIT WISE OPERATORS

BELOW ARE THE BIT-WISE OPERATORS AND THEIR NAME IN C LANGUAGE.

1. & – Bitwise AND
2. | – Bitwise OR
3. ~ – Bitwise NOT
4. ^ – XOR
5. << – Left Shift
6. >> – Right Shift

- **Exact Size Integer Types:-**
  - The following table provides the details of standard integer types with their storage sizes and value ranges −

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	8 bytes	-9223372036854775808 to 9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615

- Bitwise operators are used for manipulating a data at the bit level, also called as bit level programming. Bit-level programming mainly consists of 0 and 1. They are used in numerical computations to make the calculation process faster.

Following is the list of bitwise operators provided by 'C' programming language:

Operators	Example/Description
&& (logical AND)	(x>5)&&(y<5) It returns true when both conditions are true
|| (logical OR)	(x>=10)||(y>=10) It returns true when at-least one of the condition is true
! (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

- **Shift Operators:-**

  Bitwise Right Shift Operator in C :-

  1.It is denoted by **>>**

2. Bit Pattern of the data can be **shifted by specified number of Positions to Right.**

3. **When Data is Shifted Right , leading zero's are filled with zero**.

4. Right shift Operator is **Binary Operator** [Bi – two]

5. Binary means , **Operator that require two arguments**

- **Bitwise Left Shift Operator in C :-**

<< (left shift) Takes two numbers, left shifts the bits of the first operand, the second operand decides the number of places to shift. Or in other words left shifting an integer "x" with an integer "y" (x<<y) is equivalent to multiplying x with 2^y (2 raise to power y).

Important Points:-

- The left shift and right shift operators should not be used for negative numbers. The result of is undefined behaviour if any of the operands is a negative number. For example results of both -1 << 1 and 1 << -1 is undefined.
- If the number is shifted more than the size of integer, the behaviour is undefined. For example, 1 << 33 is undefined if integers are stored using 32 bits. See this for more details.
- The left-shift by 1 and right-shift by 1 are equivalent to multiplication and division by 2 respectively.

- **Selection & Making Decisions:-**
  - Decision making structures require that the programmer specifies one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.
  - Show below is the general form of a typical decision making structure found in most of the programming languages –

- C programming language assumes any **non-zero** and **non-null** values as **true**, and if it is either **zero** or **null**, then it is assumed as **false** value.

1. If Statement

"If statement" is the selection statement used to select course of action depending on the conditions given. Therefore programmers can use this statement to control the flow of their program.

The syntax of the if statement in C programming is:

```
if (test expression)
{
   // statements to be executed if the test expression is true
}
```

2. C if...else Statement

The if statement may have an optional else block. The syntax of the if..else statement is:

```
if (test expression) {
    // statements to be executed if the test expression is true
}
else {
    // statements to be executed if the test expression is false
}
```

Nested if...else

It is possible to include an if..else statement inside the body of another if...else statement.

➢ Logical Data and Operators:-

Following table shows all the logical operators supported by C language. Assume variable **A** holds 1 and variable **B** holds 0, then

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B) is true.

Try the following example to understand all the logical operators available in C −

```
#include <stdio.h>

main() {

   int a = 5;
   int b = 20;
   int c ;

   if ( a && b ) {
      printf("Line 1 - Condition is true\n" );
   }

   if ( a || b ) {
      printf("Line 2 - Condition is true\n" );
   }

   /* lets change the value of  a and b */
   a = 0;
   b = 10;

   if ( a && b ) {
      printf("Line 3 - Condition is true\n" );
   } else {
      printf("Line 3 - Condition is not true\n" );
   }

   if ( !(a && b) ) {
      printf("Line 4 - Condition is true\n" );
   }

}
```

```
Line 1 - Condition is true
Line 2 - Condition is true
Line 3 - Condition is not true
Line 4 - Condition is true
```

When you compile and execute the above program, it produces the following result −

```
Line 1 - Condition is true
Line 2 - Condition is true
Line 3 - Condition is not true
Line 4 - Condition is true
```

- **Two Way Selection:-**
  - The two-way selection is the basic decision statement for computers.
  - The decision is based on resolving a binary expression, and then executing a set of commands depending on whether the response was true or false.
  - C, like most contemporary programming languages, implements two-way selection with the if…else statement.

- **Multi-Way Selection:-**

  A multi-way selection statement is used to execute **at most ONE** of the choices of a set of statements presented.

  Syntax of the multi-way select statement:

  Switch (Expression)
  {
  Case Constant1, one or more statements; break;
  Case constant2, one or more statements; break;
  ……..
  [default : one or more statements;]
  }

- **More Standard Functions:-**
  - The standard functions are built-in functions. In C programming language, the standard functions are declared in header files and defined in .dll files.
  - In simple words, the standard functions can be defined as "the ready made functions defined by the system to make coding more easy".
  - The standard functions are also called as **library functions** or **pre-defined functions**.
  - in C when we use standard functions, we must include the respective header file using **#include** statement.

- For example, the function **printf()** is defined in header file **stdio.h** (Standard Input Output header file).
- When we use **printf()** in our program, we must include **stdio.h** header file using **#include<stdio.h>** statement
- C Programming Language provides the following header files with standard functions.

<ctype.h> character testing and conversion functions.
<math.h> Mathematical functions
<stdio.h> standard I/O library functions
<stdlib.h> Utility functions such as string conversion routines memory allocation routines , random number generator,etc.
<string.h> string Manipulation functions
<time.h> Time Manipulation functions

MATH.H

abs : returns the absolute value of an integer x
cos : returns the cosine of x, where x is in radians
exp: returns "e" raised to a given power
fabs: returns the absolute value of a float x
log: returns the logarithm to base e
log10: returns the logarithm to base 10
pow : returns a given number raised to another number
sin : returns the sine of x, where x is in radians
sqrt : returns the square root of x

❖ Repetition:-

➢ Concept of Loop:-

- In looping, a program executes the sequence of statements many times until the stated condition becomes false. A loop consists of two parts, a body of a loop and a control statement.
- The control statement is a combination of some conditions that direct the body of the loop to execute until the specified condition

Types of Loops

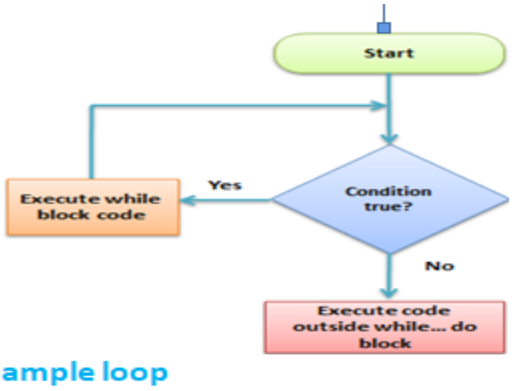
Depending upon the position of a control statement in a program, a loop is classified into two types:

1. Entry controlled loop

2. Exit controlled loop

In an **entry controlled loop,** a condition is checked before executing the body of a loop. It is also called as a pre-checking loop.

In an **exit controlled loop**, a condition is checked after executing the body of a loop. It is also called as a post-checking loop



Sample loop

The control conditions must be well defined and specified otherwise the loop will execute an infinite number of times.

'C' programming language provides us with three types of loop constructs:

1. The while loop

2. The do-while loop

3. The for loop

While Loop

A while loop is the most straightforward looping structure. The basic format of while loop is as follows:

```
While (condition )
{
Statements;
}
```

It is an entry-controlled loop. In while loop, a condition is evaluated before processing a body of the loop.

Following program illustrates a while loop:

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

```
#include<conio.h>
int main()
{
        int num=1;      //initializing the variable
        while(num<=10)          //while loop with condition
        {
                printf("%d\n",num);
                num++;          //incrementing operation
        }
        return 0;
}
```

Output:

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

Do-While loop

A do-while loop is similar to the while loop except that the condition is always executed after the body of a loop. It is also called an exit-controlled loop.

The basic format of while loop is as follows:

```
do {

 statements

} while (expression);
```

As we saw in a while loop, the body is executed if and only if the condition is true. In some cases, we have to execute a body of the loop at least once even if the condition is false. This type of operation can be achieved by using a do-while loop

In the do-while loop, the body of a loop is always executed at least once. After the body is executed, then it checks the condition. If the condition is true, then it will again execute the body of a loop otherwise control is transferred out of the loop.

The following program illustrates the working of a do-while loop:

```
#include<stdio.h>
#include<conio.h>
int main()
{
	int num=1;	//initializing the variable
	do	//do-while loop
	{
		printf("%d\n",2*num);
		num++;		//incrementing operation
	}while(num<=10);
	return 0;
}
```

Output:

```
2
4
6
8
10
12
14
16
18
20
```

For loop

A for loop is a more efficient loop structure in 'C' programming. The general structure of for loop is as follows:

```
for (initial value; condition; incrementation or decrementation )
{
  statements;
}
```

- The initial value of the for loop is performed only once.
- The condition is a Boolean expression that tests and compares the counter to a fixed value after each iteration, stopping the for loop when false is returned.
- The incrementation/decrementation increases (or decreases) the counter by a set value.

Following program illustrates the use of a simple for loop:

```
#include<stdio.h>
int main()
{
	int number;
	for(number=1;number<=10;number++) //for loop to print 1-10 numbers
	{
		printf("%d\n",number);		//to print the number
	}
	return 0;
```

}

Output:

1
2
3
4
5
6
7
8
9
10

- **Pretest and Post-test Loops:-**

a) In a pretest loop, the test condition for the execution of the repetition statement is evaluated before executing each run of the loop.

In a posttest loop, the test condition for the execution of the repetition statement is evaluated after executing each run of the loop.

The differences between pretest and posttest loops are described in the table below:

S.No.	Pretest Loops	Posttest Loops
1.	Test condition is evaluated before executing any statement within the loop.	Test condition is evaluated after executing all statements within the loop.
2.	If the test condition is false, the loop statements never execute.	Even if the test condition is false, the loop statements execute once.
3.	The `for` and `while` statements are pretest loops in C++.	The `do-while` statement is posttest loop in C++.

Initialization:-

- Initialization is the process of locating and using the defined values for variable data that is used by a computer program. For example, an operating system or application program is installed with default or user-specified values that determine certain aspects of how the system or program is to function.
- The process of the user specifying initialization values is sometimes called *configuration*.

➢ Event and Counter Controlled Loops:-

The *Event-Controlled* while loop

- In this **while** loop an action is repeated until a certain event occurs. This is by far the most frequently used form of the **while** loop.
- There are three different types of **Event-Controlled** while loops

1. **Sentinel-controlled loops** - A *sentinel* variable is initialized to a specific value. The **while** loop continues until, through some action inside the loop, the *sentinel* variable is set to a predefined *termination* value.

2. **End-of-file controlled loops** - This type of **while** loop is usually used when reading from a file. The loop terminates when the end of the file is detected. This can easily be determined by checking the **EOF()** function after each read from the file. This function will return true when the end-of-file is reached.

3. **Flag controlled loops** - A **bool** variable is defined and initialized to serve as a *flag*. The **while** loop continues until the *flag* variable value flips (true becomes false or false becomes true).

- **Count-Controlled Repetition:-**

Count-controlled repetition requires

- control variable (or loop counter)
- initial value of the control variable
- increment (or decrement) by which the control variable is modified each iteration through the loop
- condition that tests for the final value of the control variable

- A count-controlled repetition will exit after running a certain number of times. The count is kept in a variable called an index or counter.
- When the index reaches a certain value (the loop bound) the loop will end.
- Count-controlled repetition is often called definite repetition because the number of repetitions is known before the loop begins executing.

➢ Other Statements Related to Looping:-

❖ break and continue:-

- ***break*** and ***continue*** are two C/C++ statements that allow us to further control flow within and out of loops.
- ***break*** causes execution to immediately jump out of the current loop, and proceed with the code following the loop.
- ***continue*** causes the remainder of the current iteration of the loop to be skipped, and for execution to recommence with the next iteration.

- In the case of ***for*** loops, the incrementation step will be executed next, followed by the condition test to start the next loop iteration.
- In the case of ***while*** and ***do-while*** loops, execution jumps to the next loop condition test.

❖ **Infinite Loops:-**

- Infinite loops are loops that repeat forever without stopping.
- Usually they are caused by some sort of error, such as the following example in which the wrong variable is incremented:

```
int i, j;
for( i = 0; i < 5; j++ )
    printf( "i = %d\n", i );
printf( "This line will never execute\n" );
```

❖ **Nested Loops**

The code inside a loop can be any valid C code, including other loops.

Any kind of loop can be nested inside of any other kind of loop.

➢ Looping Applications:-

Why do we use looping in programming?

Because we want to repeat something:

- Count from 1 to 10.
- Go through all the words in a dictionary to see whether they're palindromes.
- For each customer that has an outstanding balance, send out an email reminder that payment is due.
- For each directory under this one, find music files and add then to the list of known music.

➢ **The Calculator Program**:-

/*C program to design calculator with basic operations using switch.*/

```
#include <stdio.h>
 int main()
{
   int num1,num2;
   float result;
   char ch;    //to store operator choice
```

```
    printf("Enter first number: ");
   scanf("%d",&num1);
   printf("Enter second number: ");
   scanf("%d",&num2);
     printf("Choose operation to perform (+,-,*,/,%): ");
   scanf(" %c",&ch);
    result=0;
 switch(ch)
   {
      case '+':
         result=num1+num2;
         break;

      case '-':
         result=num1-num2;
         break;

      case '*':
         result=num1*num2;
         break;

      case '/':
         result=(float)num1/(float)num2;
         break;

      case '%':
         result=num1%num2;
         break;
      default:
         printf("Invalid operation.\n");
   }

   printf("Result: %d %c %d = %f\n",num1,ch,num2,result);
   return 0;
}
```

Output

```
Enter first number: 10
Enter second number: 20
Choose operation to perform (+,-,*,/,%): +
Result: 10 + 20 = 30.000000

Enter first number: 10
Enter second number: 3
Choose operation to perform (+,-,*,/,%): /
Result: 10 / 3 = 3.333333

Enter first number: 10
```

```
Enter second number: 3
Choose operation to perform (+,-,*,/,%): >
Invalid operation.
Result: 10 > 3 = 0.000000
```

The End

PROGRAMMING FOR PROBLEM SOLVING USING C
UNIT-III

Arrays: Concepts, Using Array in C, Array Application, Two Dimensional Arrays, Multidimensional Arrays, Programming Example – Calculate Averages, Tips and Common Programming Errors, Key Terms, Summary, Practice Set.
Strings: String Concepts, C String, String Input / Output Functions, Arrays of Strings, String Manipulation Functions String/ Data Conversion, A Programming Example – Morse Code, Tips and Common Programming Errors, Key Terms, Summary, Practice Set.
Enumerated, Structure, and Union: The Type Definition (Type def), Enumerated Types, Structure, Unions, Programming Application, Tips and Common Programming Errors, Key Terms, Summary, Practice Set.

➢ Arrays:-

An array is a group of related data items that share a common name.
Ex:-Students

- The complete set of students are represented using an array name students.
- A particular value is indicated by writing a number called index number or subscript in brackets after array name.
- The complete set of value is referred to as an array, the individual values are called elements.

➢ Using Array in C:-

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

Since arrays provide an easy way to represent data, it is classified amongst the data structures in C. Other data structures in c are structure, lists, queues, trees etc. Array can be used to represent not only simple list of data but also table of data in two or three dimensions.

➢ Array Application:-

In c programming language, arrays are used in wide range of applications. Few of them are as follows.

❖ Arrays are used to Store List of values

In c programming language, single dimensional arrays are used to store list of values of same datatype. In other words, single dimensional arrays are used to store a row of values. In single dimensional array data is stored in linear form

❖ Arrays are used to Perform Matrix Operations

We use two dimensional arrays to create matrix. We can perform various operations on matrices using two dimensional arrays.

❖ **Arrays are used to Perform Matrix Operations**

We use two dimensional arrays to create matrix. We can perform various operations on matrices using two dimensional arrays.

❖ **Arrays are used to implement Search Algorithms**

We use single dimensional arrays to implement search algorihtms like ...

1. **Linear search**
2. **Binary search**

❖ **Arrays are used to implement Sorting Algorithms**

We use single dimensional arrays to implement sorting algorithms like ...

1. Insertion sort
2. Bubble sort
3. Selection sort
4. Quick sort
5. Merge sort

➢ **ONE – DIMENSIONAL ARRAYS :-**

A list of items can be given one variable index is called single subscripted variable or a one-dimensional array.

The subscript value starts from 0. If we want 5 elements the declaration will be

int number[5];

The elements will be number[0], number[1], number[2], number[3], number[4]
There will not be number[5]

➢ **Declaration of One - Dimensional Arrays :**

Type variable – name [sizes];

Type – data type of all elements Ex: int, float etc.,
Variable – name – is an identifier
Size – is the maximum no of elements that can be stored

Ex:- float avg[50]

This array is of type float. Its name is avg. and it can contains 50 elements only. The range starting from 0 – 49 elements.

- **Initialization of Arrays :-**

  Initialization of elements of arrays can be done in same way as ordinary variables are done when they are declared.

  Type array name[size] = {List of Value};

  Ex:- int number[3]={0,0,0};

  If the number of values in the list is less than number of elements then only that elements will be initialized. The remaining elements will be set to zero automatically.

- **TWO – DIMENSIONAL ARRAYS:-**

  To store tables we need two dimensional arrays. Each table consists of rows and columns. Two dimensional arrays are declare as

  type array name [row-size][col-size];

/* Write a program Showing 2-DIMENSIONAL ARRAY */
/* SHOWING MULTIPLICATION TABLE */

```
#include<stdio.h>
#include<math.h>
#define ROWS 5
#define COLS 5
main()
{
int row,cols,prod[ROWS][COLS];
int i,j;
printf("Multiplication table");
for(j=1;j< =COLS;j++)
printf("%d",j);
for(i=0;i<ROWS;i++)
{
row = i+1;
printf("%2d|",row);
for(j=1;j < = COLS;j++)
{
```

```
COLS=j;
prod[i][j]= row * cols;
printf("%4d",prod*i+*j+);
}
}
}
```

- **INITIALIZING TWO DIMENSIONAL ARRAYS:-**
  They can be initialized by following their declaration with a list of initial values enclosed in braces.

  Ex:- int table[2][3] = {0,0,0,1,1,1};

  Initializes the elements of first row to zero and second row to one. The initialization is done by row by row. The above statement can be written as
  int table[2][3] = {{0,0,0},{1,1,1}};

When all elements are to be initialized to zero, following short-cut method may be used.

int m[3][5] = {{0},{0},{0}};

- **Multidimensional Arrays:-**

  C allows for arrays of two or more dimensions. A two-dimensional (2D) array is an array of arrays. A three-dimensional (3D) array is an array of arrays of arrays.

  In C programming an array can have two, three, or even ten or more dimensions. The maximum dimensions a C program can have depends on which compiler is being used.

- **Declare a Multidimensional Array in C:-**
  A multidimensional array is declared using the following syntax:

  type array_name[d1][d2][d3][d4]……[dn];

  Where each **d** is a dimension, and **dn** is the size of final dimension.

Examples:

1. int table[5][5][20];
2. float arr[5][6][5][6][5];

In Example 1:

- **int** designates the array type integer.
- **table** is the name of our 3D array.
- Our array can hold 500 integer-type elements. This number is reached by multiplying the value of each dimension. In this case: **5x5x20=500**.

In Example 2:

- Array **arr** is a five-dimensional array.
- It can hold 4500 floating-point elements (**5x6x5x6x5=4500**).

➢ Initializing a 3D Array in C:-

Like any other variable or array, a 3D array can be initialized at the time of compilation. By default, in C, an uninitialized 3D array contains “garbage” values, not valid for the intended use.

Let’s see a complete example on how to initialize a 3D array:-

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i, j, k;
int arr[3][3][3]=
      {
         {
         {11, 12, 13},
         {14, 15, 16},
         {17, 18, 19}
         },
         {
         {21, 22, 23},
         {24, 25, 26},
         {27, 28, 29}
         },
         {
         {31, 32, 33},
         {34, 35, 36},
         {37, 38, 39}
         },
      };
clrscr();
printf(":::3D Array Elements:::\n\n");
for(i=0;i<3;i++)
```

```
{
   for(j=0;j<3;j++)
   {
      for(k=0;k<3;k++)
      {
      printf("%d\t",arr[i][j][k]);
      }
      printf("\n");
   }
   printf("\n");
}
getch();
}
```

➢ Programming Example – Calculate Averages:-

This program takes n number of element from user (where, n is specified by user), stores data in an array and calculates the average of those numbers.

Source Code to Calculate Average Using Arrays:-

```
#include <stdio.h>

int main()
{
   int n, i;
   float num[100], sum = 0.0, average;

   printf("Enter the numbers of elements: ");
   scanf("%d", &n);

   while (n > 100 || n <= 0)
   {
      printf("Error! number should in range of (1 to 100).\n");
      printf("Enter the number again: ");
      scanf("%d", &n);
   }

   for(i = 0; i < n; ++i)
   {
      printf("%d. Enter number: ", i+1);
      scanf("%f", &num[i]);
      sum += num[i];
   }

   average = sum / n;
   printf("Average = %.2f", average);

   return 0;
```

}

Output

```
Enter the numbers of elements: 6
1. Enter number: 45.3
2. Enter number: 67.5
3. Enter number: -45.6
4. Enter number: 20.34
5. Enter number: 33
6. Enter number: 45.6
Average = 27.69
```

➢ Strings:-

A String is an array of characters. Any group of characters (except double quote sign)defined between double quotes is a constant string.

Ex: “C is a great programming language”.

If we want to include double quotes.

Ex: “\”C is great \” is norm of programmers “.

➢ Declaring and initializing strings :-

A string variable is any valid C variable name and is always declared as an array.

char string name [size];

size determines number of characters in the string name. When the compiler assigns a character string to a character array, it automatically supplies a null character (‘\0’) at end of String.

Therefore, size should be equal to maximum number of character in String plus one.

There are two ways to declare a string in c language.

1. By char array
2. By string literal

1. char city*10+= ,‘N’,’E’,’W’,’ ‘,’Y’,’O’,’R’,’K’,’\0’-;

2. char city*10+= “NEW YORK’;

C also permits us to initializing a String without specifying size.

Ex:- char Strings*+= ,'G','O','O','D','\0'-;

➢ String Input / Output Functions:-

- C provides two basic ways to read and write strings
- First we can read and write strings with the formatted input/output functions,scanf/fscanf and prinf/fprinf.
- Second we can use a special set of strin only functions ,get string(gets/fgets)and put string(puts/fputs).

Formatted string Input/Output:

- Formatted String Input:scanf/fscanf:
- **int fscanf(FILE **stream*, const char **format*, ...);**
- **int scanf(const char **format*, ...);**
- The ..scanf functions provide a means to input formatted information from a stream.
- **fscanf** reads formatted input from a stream
- **scanf** reads formatted input stdin

These functions take input in a manner that is specified by the format argument and store each input field into the following arguments in a left to right fashion.

String Input/Output

In addition to the Formatted string functions,C has two sets of string functions that read and write strings without reformatting any data.These functions convert text file lines to strings and strings to text file lines

gets():

Declaration:

```
char *gets(char *str);
```

Reads a line from **stdin** and stores it into the string pointed to by *str*. It stops when either the newline character is read or when the end-of-file is reached, whichever comes first. The newline character is not copied to the string. A null character is appended to the end of the string.

puts:

Declaration:

`int puts(const char *`*`str`***`);`**

Writes a string to **`stdout`** up to but not including the null character. A newline character is appended to the output.

On success a nonnegative value is returned. On error **`EOF`** is returned.

➢ STRING HANDLING/MANIPULATION FUNCTIONS:-

strcat( )	Concatenates two Strings
strcmp( )	Compares two Strings
strcpy( )	Copies one String Over another
strlen( )	Finds length of String

✓ strcat() function:

This function adds two strings together.

Syntax: char *strcat(const char *string1, char *string2);

strcat(string1,string2);
string1 = VERY
string2 = FOOLISH
strcat(string1,string2);
string1=VERY FOOLISH
string2 = FOOLISH

Strncat: Append n characters from string2 to stringl.
char *strncat(const char *string1, char *string2, size_t n);

✓ strcmp() function :

This function compares two strings identified by arguments and has a value 0 if they are equal. If they are not, it has the numeric difference between the first non-matching characters in the Strings.

Syntax: int strcmp (const char *string1,const char *string2);

strcmp(string1,string2);

Ex:- strcmp(name1,name2);
strcmp(name1,"John");
strcmp("ROM","Ram");

Strncmp: Compare first n characters of two strings.

int strncmp(const char *string1, char *string2, size_t n);

✓ strcpy() function :

It works almost as a string assignment operators. It takes the form

Syntax: char *strcpy(const char *string1,const char *string2);

strcpy(string1,string2);
string2 can be array or a constant.

Strncpy: Copy first n characters of string2 to stringl .

char *strncpy(const char *string1,const char *string2, size_t n);

✓ strlen() function :

Counts and returns the number of characters in a string.

Syntax:int strlen(const char *string);

n= strlen(string);

n→ integer variable which receives the value of length of string.

➢ String/ Data Conversion:-

✓ **atof :-** convert a string to a double precision number.

Usage

```
Double_Type atof (String_Type s)
```

✓ **atoi:-** convert a string to an integer

Usage

```
Int_Type atoi (String_Type str)
```

✓ **atol**:- convert a string to an long integer.

Usage

```
Long_Type atol (String_Type str)
```

✓ **char**:- convert a character code to a string.

Usage

```
String_Type char (Integer_Type c)
```

✓ **integer :-** Convert a string to an integer

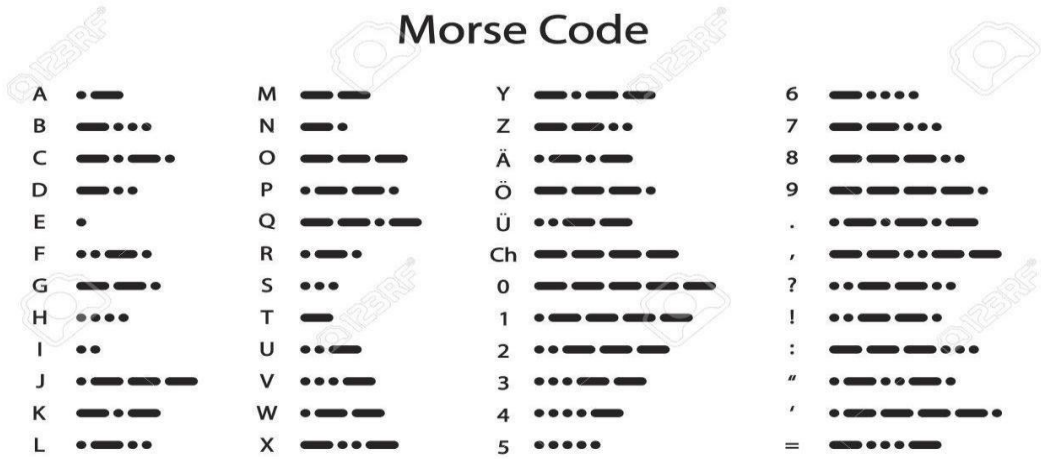
Usage

```
Integer_Type integer (String_Type s)
```

➢ **A Programming Example – Morse Code:-**

Morse code is a method of transmitting text information as a series of on-off tones, lights, or clicks that can be directly understood by a skilled listener or observer without special equipment.

It is named for Samuel F. B. Morse, an inventor of the telegraph.



We're gonna make an array with these values. Actually , just for the alphabet (a through z). The rest don't really matter that much.

So this is our array of strings:

```
char *u[36] = {".-","-...","-.-.","-..",".","..-.","--.","....","..",".---
","-.-","-..","--","-.","---",".--.","--.-",".-.","...","-","..-","...-",".--
","-..-","-.--","--..","-----",".----","..---","...--","....-",".....","-....","-
-...","---..","----."};
```

Program:-

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

int main()
{
        int i;
        char input[255], morse['o ---', '--- o o o', '--- o --- o', '--- o o', 'o',
                        'o o --- o', '--- --- o', 'o o o o', 'o o', 'o ----------',
                                                '--- o ---', 'o --- o o', '--- ---', '--- o', ' ----------',
                                                'o --- --- o', '--- --- o ---', 'o --- o', 'o o o', '---',
                                                'o o ---', 'o o o ---', 'o --- ---', '--- o o ---', '--- o ---
---',
                                                ' ------ o o']

        printf("Enter your string: ");

        gets(string);

        for (i = 0; input[i] != '\0'; i++)
        {
                if ()

        }

        printf(" Your string in morse is: ");

        return 0;
}
```

➢ The Type Definition (Type def):-

- The C programming language provides a keyword called **typedef**, which you can use to give a type a new name.
- Following is an example to define a term **BYTE** for one-byte numbers –

```
typedef unsigned char BYTE;
```

- After this type definition, the identifier BYTE can be used as an abbreviation for the type unsigned char, for example..

```
BYTE  b1, b2;
```

- By convention, uppercase letters are used for these definitions to remind the user that the type name is really a symbolic abbreviation, but you can use lowercase, as follows –

```
typedef unsigned char byte;
```

- You can use **typedef** to give a name to your user defined data types as well. For example, you can use typedef with structure to define a new data type and then use that data type to define structure variables directly as follows –

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

typedef struct Books {
   char title[50];
   char author[50];
   char subject[100];
   int book_id;
} Book;

int main( ) {
Book book;
   strcpy( book.title, "C Programming");
   strcpy( book.author, "Nuha Ali");
   strcpy( book.subject, "C Programming Tutorial");
   book.book_id = 6495407;

   printf( "Book title : %s\n", book.title);
   printf( "Book author : %s\n", book.author);
   printf( "Book subject : %s\n", book.subject);
   printf( "Book book_id : %d\n", book.book_id);

   return 0;
}
```

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

```
Book  title : C Programming
Book  author : Nuha Ali
Book  subject : C Programming Tutorial
Book  book_id : 6495407
```

➢ Enumerated Types:-

- Enumeration (or enum) is a user defined data type in C. It is mainly used to assign names to integral constants, the names make a program easy to read and maintain.

Here is the syntax of enum in C language

```
enum enum_name{const1, const2, ......... };
```

- The enum keyword is also used to define the variables of enum type. There are two ways to define the variables of enum type as follows.

1. `enum week{sunday, monday, tuesday, wednesday, thursday, friday, saturday};`
2. `enum week day;`

Example:-

```
#include<stdio.h>
enum week{Mon=10, Tue, Wed, Thur, Fri=10, Sat=16, Sun};
enum day{Mond, Tues, Wedn, Thurs, Frid=18, Satu=11, Sund};
int main() {
  printf("The value of enum week: %d\t%d\t%d\t%d\t%d\t%d\t%d\n\n",Mon , Tue, Wed, Thur, Fri, Sat, Sun);
  printf("The default value of enum day: %d\t%d\t%d\t%d\t%d\t%d\t%d",Mond , Tues, Wedn, Thurs, Frid, Satu, Sund);
  return 0;
}
```

Output:-

```
The value of enum week: 10 11    12    13    10    16    17
The default value of enum day: 0 1     2     3     18    11    12
```

- In the above program, two enums are declared as week and day outside the main() function. In the main() function, the values of enum elements are printed.

➢ Structure:-

- Arrays allow to define type of variables that can hold several data items of the same kind. Similarly **structure** is another user defined data type available in C that allows to combine data items of different kinds.
- Structures are used to represent a record. Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book –

Title
Author
Subject
Book ID

➢ Defining a Structure:-

- To define a structure, you must use the **struct** statement. The struct statement defines a new data type, with more than one member. The format of the struct statement is as follows –

```
struct [structure tag] {
member definition;
  member definition;
...
  member definition;
} [one or more structure variables];
```

- The **structure tag** is optional and each member definition is a normal variable definition, such as int i; or float f; or any other valid variable definition.

Accessing Structure Members:-

- To access any member of a structure, we use the **member access operator (.)**. The member access operator is coded as a period between the structure variable name and the structure

member that we wish to access. You would use the keyword **struct** to define variables of structure type.

- The following example shows how to use a structure in a program −

```
#include  <stdio.h>
#include <string.h>
struct Books {
  char  title[50];
  char author[50];
  char subject[100];
  int  book_id;
};
 int main( ) {
  struct Books Book1;       /* Declare Book1 of type Book */
  struct Books Book2;       /* Declare Book2 of type Book */

  /* book 1 specification */
  strcpy( Book1.title, "C Programming");
  strcpy( Book1.author, "Nuha Ali");
  strcpy( Book1.subject, "C Programming Tutorial");
  Book1.book_id = 6495407;

  /* book 2 specification */
  strcpy( Book2.title, "Telecom Billing");
  strcpy( Book2.author, "Zara Ali");
  strcpy( Book2.subject, "Telecom Billing Tutorial");
  Book2.book_id = 6495700;

  /* print Book1 info */
  printf( "Book 1 title : %s\n", Book1.title);
  printf( "Book 1 author : %s\n", Book1.author);
  printf( "Book 1 subject : %s\n", Book1.subject);
  printf( "Book 1 book_id : %d\n", Book1.book_id);

  /* print Book2 info */
  printf( "Book 2 title : %s\n", Book2.title);
  printf( "Book 2 author : %s\n", Book2.author);
  printf( "Book 2 subject : %s\n", Book2.subject);
  printf( "Book 2 book_id : %d\n", Book2.book_id);
  return 0;  }
```

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

```
Book 1 title : C Programming
Book 1 author : Nuha Ali
Book 1 subject : C Programming Tutorial
Book 1 book_id : 6495407
Book 2 title : Telecom Billing
Book 2 author : Zara Ali
Book 2 subject : Telecom Billing Tutorial
Book 2 book_id : 6495700
```

Structures as Function Arguments:-

You can pass a structure as a function argument in the same way as you pass any other variable or pointer.

➢ Unions:-

- A **union** is a special data type available in C that allows to store different data types in the same memory location.
- You can define a union with many members, but only one member can contain a value at any given time.
- Unions provide an efficient way of using the same memory location for multiple-purpose.

Defining a Union:-

- To define a union, you must use the **union** statement in the same way as you did while defining a structure.
- The union statement defines a new data type with more than one member for your program.
- The format of the union statement is as follows −

```
union [union tag] {
  member definition;
  member definition;
  ...
  member definition;
} [one or more union variables];
```

The **union tag** is optional and each member definition is a normal variable definition, such as int i; or float f; or any other valid variable definition.

```
union Data {
  int i;
  float f;
  char str[20];
} data;
```

- Now, a variable of **Data** type can store an integer, a floating-point number, or a string of characters.
- It means a single variable, i.e., same memory location, can be used to store multiple types of data.
- You can use any built-in or user defined data types inside a union based on your requirement.
- The memory occupied by a union will be large enough to hold the largest member of the union. For example, in the above example, Data type will occupy 20 bytes of memory space because this is the maximum space which can be occupied by a character string.
- The following example displays the total memory size occupied by the above union –

```
#include  <stdio.h>
#include <string.h>
union Data {
  int i;
  float f;
  char str[20];
};
 int main( ) {
 union Data data;
  printf( "Memory size occupied by data : %d\n", sizeof(data));
 return 0;
}
```

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

```
Memory size occupied by data : 20
```

Accessing Union Members:-

- To access any member of a union, we use the **member access operator (.)**. The member access operator is coded as a period

between the union variable name and the union member that we wish to access.

- You would use the keyword **union** to define variables of union type. The following example shows how to use unions in a program –

```
#include  <stdio.h>
#include <string.h>
union Data {
  int i;
  float f;
  char str[20];
};
 int main( ) {
 union Data data;
 data.i = 10;
  data.f = 220.5;
  strcpy( data.str, "C Programming");
   printf( "data.i : %d\n", data.i);
  printf( "data.f : %f\n", data.f);
 printf( "data.str : %s\n", data.str);
 return 0;
}
```

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

```
data.i : 1917853763
data.f : 4122360580327794860452759994368.000000
data.str : C Programming
```

➢ **Programming Application:-**

Mainly C Language is used for Develop Desktop application and system software. Some application of C language are given below.

- C programming language can be used to design the system software like operating system and Compiler.
- To develop application software like database and spread sheets.
- For Develop Graphical related application like computer and mobile games.

- To evaluate any kind of mathematical equation use c language.
- C programming language can be used to design the compilers.
- UNIX Kernal is completely developed in C Language.
- For Creating Compilers of different Languages which can take input from other language and convert it into lower level machine dependent language.
- C programming language can be used to design Operating System.
- C programming language can be used to design Network Devices.

Pointers: Introduction, Pointers to pointers, Compatibility, L value and R value, Tips and Common Programming Errors, Key Terms, Summary, Practice Set.
Pointer Applications: Arrays, and Pointers, Pointer Arithmetic and Arrays, Memory Allocation Function, Array of Pointers, Programming Application, Tips and Common Programming Errors, Key Terms, Summary, Practice Set.
Processor Commands: Processor Commands, Tips and Common Programming Errors, Key Terms, Summary, Practice Set.

➢ Introduction:-

- One of the powerful features of C is ability to access the memory variables by their memory address.
- This can be done by using Pointers. The real power of C lies in the proper use of Pointers.
- A pointer is a variable that can store an address of a variable (i.e., 112300).We say that a pointer points to a variable that is stored at that address.
- A pointer itself usually occupies 4 bytes of memory (then it can address cells from 0 to 232-1).

Advantages of Pointers:-

1. A pointer enables us to access a variable that is defined out side the function.
2. Pointers are more efficient in handling the data tables.
3. Pointers reduce the length and complexity of a program.
4. They increase the execution speed.

Definition :-

A variable that holds a physical memory address is called a pointer variable or Pointer

Declaration :

Datatype * Variable-name;

```
Eg:-    int *ad; /* pointer to int */
        char *s; /* pointer to char */
        float *fp; /* pointer to float */
        char **s; /* pointer to variable that is a pointer to char */
```

- A pointer is a variable that contains an address which is a location of another variable in memory.

Consider the Statement

p=&i;

Here „&" is called address of a variable.
'p' contains the address of a variable i.

The operator & returns the memory address of variable on which it is operated, this is called Referencing.

The * operator is called an indirection operator or dereferencing operator which is used to display the contents of the Pointer Variable.

Consider the following Statements :

```
int *p,x;
x =5;
p= &x;
```

Assume that x is stored at the memory address 2000. Then the output for the following printf statements is :

	Output
Printf("The Value of x is %d",x);	5
Printf("The Address of x is %u",&x);	2000
Printf("The Address of x is %u",p);	2000
Printf("The Value of x is %d",*p);	5
Printf("The Value of x is %d",*(&x));	5

➢ Pointers to pointers:-

- A pointer to a pointer is a form of multiple indirection, or a chain of pointers. Normally, a pointer contains the address of a variable.
- When we define a pointer to a pointer, the first pointer contains the address of the second pointer, which points to the location that contains the actual value as shown below.

- A variable that is a pointer to a pointer must be declared as such.

- This is done by placing an additional asterisk in front of its name. For example, the following declaration declares a pointer to a pointer of type int –

```
int **var;
```

➢ **Compatibility pointer:-**

The rules for assigning one pointer to another are tighter than the rules for numeric types.

For example, you can assign an int value to a double variable without using a type conversion, but you can't do the same for pointers to these two types. Let's see a simple C program to exemplify this.

```
*
 * ptr_compatibility.c -- program illustrates concept of pointer
 * compatibility
 */
#include <stdio.h>

int main(void)
{
    int n = 5;
    long double x;

    int *pi = &n;
    long double *pld = &x;

    x = n;     /* implicit type conversion */
    pld = pi;  /* compile-time error: assigning pointer-to-int to */
               /* pointer-to-long-double */

    return 0;
}
```

➢ **L value and R value:-**

L-value: "l-value" refers to memory location which identifies an object. l-value may appear as either left hand or right hand side of an assignment operator(=). l-value often represents as identifier.

Expressions referring to modifiable locations are called "**modifiable l-values**". A modifiable l-value cannot have an array type, an incomplete type, or a type with the **const** attribute

In C, the concept was renamed as **"locator value"**, and referred to expressions that locate (designate) objects.

The l-value is one of the following:

1. he name of the variable of any type i.e, an identifier of integral, floating, pointer, structure, or union type.
2. A subscript ([ ]) expression that does not evaluate to an array.
3. A unary-indirection (*) expression that does not refer to an array
4. An l-value expression in parentheses.
5. A **const** object (a nonmodifiable l-value).
6. The result of indirection through a pointer, provided that it isn't a function pointer.
7. The result of member access through pointer(-> or .)

R-value: r-value" refers to data value that is stored at some address in memory. A r-value is an expression that can't have a value assigned to it which means r-value can appear on right but not on left hand side of an assignment operator(=).

Note: The unary & (address-of) operator requires an lvalue as its operand. That is, &n is a valid expression only if n is an lvalue.

- **Arrays:-**
  - An array is defined as the collection of similar type of data items stored at contiguous memory locations.
  - Arrays are the derived data type in C programming language which can store the primitive type of data such as int, char, double, float, etc.
  - It also has the capability to store the collection of derived data types, such as pointers, structure, etc.
  - The array is the simplest data structure where each data element can be randomly accessed by using its index number.
- **Pointer Arithmetic in C:-**

Pointer Arithmetic in C

- We can perform arithmetic operations on the pointers like addition, subtraction, etc.
- However, as we know that pointer contains the address, the result of an arithmetic operation performed on the

pointer will also be a pointer if the other operand is of type integer.

- In pointer-from-pointer subtraction, the result will be an integer value.
- Following arithmetic operations are possible on the pointer in C language:

  - Increment
  - Decrement
  - Addition
  - Subtraction
  - Comparison

Incrementing Pointer in C:-

- If we increment a pointer by 1, the pointer will start pointing to the immediate next location.
- This is somewhat different from the general arithmetic since the value of the pointer will get increased by the size of the data type to which the pointer is pointing.

The Rule to increment the pointer is given below:

new_address= current_address + i * size_of(data type)

Where i is the number by which the pointer get increased.

Decrementing Pointer in C

- Like increment, we can decrement a pointer variable. If we decrement a pointer, it will start pointing to the previous location.
- The formula of decrementing the pointer is given below:

new_address= current_address - i * size_of(data type)

C Pointer Addition

We can add a value to the pointer variable. The formula of adding value to pointer is given below:

new_address= current_address + (number * size_of(data type))

C Pointer Subtraction

Like pointer addition, we can subtract a value from the pointer variable. Subtracting any number from a pointer will give an address. The formula of subtracting value from the pointer variable is given below:

new_address= current_address - (number * size_of(data type))

- **Memory Allocation Function:-**
  - The concept of **dynamic memory allocation in c language** *enables the C programmer to allocate memory at runtime*.
  - Dynamic memory allocation in c language is possible by 4 functions of stdlib.h header file.

1. malloc()
2. calloc()
3. realloc()
4. free()

Before learning above functions, let's understand the difference between static memory allocation and dynamic memory allocation.

static memory allocation	dynamic memory allocation
memory is allocated at compile time.	memory is allocated at run time.
memory can't be increased while executing program.	memory can be increased while executing program.
used in array.	used in linked list.

Now let's have a quick look at the methods used for dynamic memory allocation.

malloc()	allocates single block of requested memory.
calloc()	allocates multiple block of requested memory.
realloc()	reallocates the memory occupied by malloc() or calloc() functions.
free()	frees the dynamically allocated memory.

➢ Array of Pointers:-

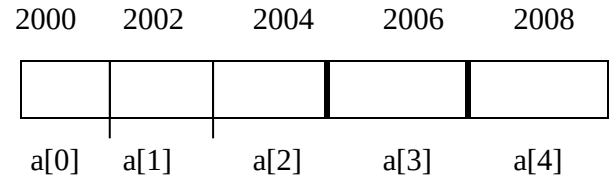
In computer programming, an array of pointers is an indexed set of variables in which the variables are pointers (a reference to a location in memory).

- Pointers are an important tool in computer science for creating, using, and destroying all types of data structures.
- An array of pointers is useful for the same reason that all arrays are useful: it allows you to numerically index a large set of variables.
- Below is an array of pointers in C that sets each pointer in one array to point to an integer in another and then print the values of the integers by dereferencing the pointers.

POINTERS WITH ARRAYS:-

When an array is declared, elements of array are stored in contiguous locations. The address of the first element of an array is called its base address.

Consider the array



The name of the array is called its base address.

i.e., a and k& a[20] are equal

Now both a and a[0] points to location 2000. If we declare p as an integer pointer, then we can make the pointer P to point to the array a by following assignment.

P = a;

We can access every value of array a by moving P from one element to another.

i.e., P points to 0th element
P+1 points to 1st element
P+2 points to 2nd element
P+3 points to 3rd element
P +4 points to 4th element

- **Programming Application:-**

APPLICATIONS OF C LANGUAGE

1. C language is used for **creating computer applications**
2. Used in writing Embedded software
3. Firmware for various electronics, industrial and communications products which use micro-controllers.
4. It is also used in developing verification software, test code, simulators etc. for various applications and hardware products.
5. For **Creating Compiler** of different Languages which can take input from other language and convert it into lower level machine dependent language.
6. C is used to implement different Operating System Operations. UNIX kernel is completely developed in C Language.

Processor Commands:-

- The **C Preprocessor** is not a part of the compiler, but is a separate step in the compilation process.
- In simple terms, a C Preprocessor is just a text substitution tool and it instructs the compiler to do required pre-processing before the actual compilation.
- We'll refer to the C Preprocessor as CPP.
- All preprocessor commands begin with a hash symbol (#).
- It must be the first nonblank character, and for readability, a preprocessor directive should begin in the first column.
- The following section lists down all the important preprocessor directives –

Sr.No.	Directive & Description
1	#define Substitutes a preprocessor macro.
2	#include Inserts a particular header from another file.
3	#undef Undefines a preprocessor macro.
4	#ifdef Returns true if this macro is defined.
5	#ifndef Returns true if this macro is not defined.
6	#if Tests if a compile time condition is true.
7	#else The alternative for #if.
8	#elif #else and #if in one statement.
9	#endif Ends preprocessor conditional.
10	#error Prints error message on stderr.
11	#pragma Issues special commands to the compiler, using a standardized method.

PROGRAMMING FOR PROBLEM SOLVING USING C
UNIT-V

Text Input / Output: Files, Streams, Standard Library Input / Output Functions, Formatting Input / Output Functions, Character Input / Output Functions, Tips and Common Programming Errors, Key Terms, Summary, Practice Set.
Binary Input / Output: Text versus Binary Streams, Standard Library, Functions for Files, Converting File Type, Tips and Common Programming Errors, Key Terms, Summary, Practice Set.
Functions: Designing, Structured Programs, Function in C, User Defined Functions, Inter-Function Communication, Standard Functions, Passing Array to Functions, Passing Pointers to Functions, Recursion, Passing an Array to Function, Tips and Common Programming Errors, Key Terms, Summary, Practice Set.

➢ Text Input / Output:

- **Files: -**
  - A file represents a sequence of bytes on the disk where a group of related data is stored.
  - File is created for permanent storage of data.
  - It is a ready-made structure.
  - You can use the fopen( ) function to create a new file or to open an existing file.
  - To close a file, use the fclose( ) function.
  - The function fputc() writes the character value of the argument c to the output stream referenced by fp.
  - The fgetc() function reads a character from the input file referenced by fp.

- **Streams:-**
  - In C all input and output is done with streams.
  - Stream is nothing but the sequence of bytes of data.
  - A sequence of bytes flowing into program is called input stream.
  - A sequence of bytes flowing out of the program is called output stream.
  - Use of Stream make I/O machine independent.

- **Standard Library I/O Functions: -**

- The standard input and output library is stdio.h, and you will find that you include this library in almost every program you write.
- It allows printing to the screen and collecting input from the user. The functions you will use the most include:
  - printf() is output to the screen
  - scanf() is read input from the screen
  - getchar() is return characters typed on screen
  - putchar() is output a single character to the screen
  - fopen() is open a file, and
  - fclose() is close a file

- **Formatting Input /Output Functions:-**
  - **Formatted Input**

    The function scanf() is used for formatted input from standard input and provides many of the conversion facilities of the function printf().

    Syntax

    scanf (format, num1, num2,……);
    The function scanf() reads and converts characters from the standards input depending on the format specification string and stores the input in memory locations represented by the other arguments (num1, num2,….).

    For Example:

    scanf(“ %c %d”,&Name, &Roll No);

- **Character Input / Output Functions:-**

Character input functions is used to input a single character.

All these functions are available in 'stdio.h' header file.

getch(): Use to input single character at a time. But it will not display input character. get stands for input, ch stands for character.
Syntax: char a = getch();

Character Output Functions**:-**
putch(): use to print a single character.
Syntax: putch(a);

putchar(): use to print a single character.
Syntax: putchar(a);

➢ Binary Input / Output:-

Text versus Binary Streams:-

Text:-

A text stream consists of one or more lines of text that can be written to a text-oriented display so that they can be read.

When reading from a text stream, the program reads an *NL* (newline) at the end of each line.

Writing to a text stream, the program writes an *NL* to signal the end of a line.

To match differing conventions among target environments for representing text in files, the library functions can alter the number and representations of characters transmitted between the program and a text stream.

Binary Stream:-

A binary stream consists of one or more bytes of arbitrary information.

You can write the value stored in an arbitrary object to a (byte-oriented) binary stream and read exactly what was stored in the object when you wrote it.

The library functions do not alter the bytes you transmit between the program and a binary stream.

They can, however, append an arbitrary number of null bytes to the file that you write with a binary stream.

The program must deal with these additional null bytes at the end of any binary stream.

- **Standard Library**:-

C Standard library functions or simply C Library functions are inbuilt functions in C programming.

The prototype and data definitions of these functions are present in their respective header files. To use these functions we need to include the header file in our program. For example,

If you want to use the printf() function, the header file <stdio.h> should be included.

```
#include<stdio.h>

int main()

{

printf("Catch me if you can.");

}
```

If you try to use printf() without including the stdio.h header file, you will get an error.

Advantages of Using C library functions

1. They work

One of the most important reasons you should use library functions is simply because they work. These functions have gone through multiple rigorous testing and are easy to use.

2. The functions are optimized for performance

Since, the functions are "standard library" functions, a dedicated group of developers constantly make them better. In the process, they are able to create the most efficient code optimized for maximum performance.

3. It saves considerable development time

Since the general functions like printing to a screen, calculating the square root, and many more are already written. You shouldn't worry about creating them once again.

4. The functions are portable

With ever-changing real-world needs, your application is expected to work every time, everywhere. And, these library functions help you in that they do the same thing on every computer.

Example: Square root using sqrt() function

Suppose, you want to find the square root of a number.

To can compute the square root of a number, you can use the sqrt() library function. The function is defined in the math.h header file.

```
#include<stdio.h>
#include<math.h>
int main()
{
floatnum, root;
printf("Enter a number: ");
scanf("%f", &num);

// Computes the square root of num and stores in root.

root = sqrt(num);

printf("Square root of %.2f = %.2f", num, root);
return0;
}
```

When you run the program, the output will be:

Enter a number: 12
Square root of 12.00 = 3.46

Library Functions in Different Header Files

C Header Files	
<assert.h>	Program assertion functions
<ctype.h>	Character type functions
<locale.h>	Localization functions
<math.h>	Mathematics functions
<setjmp.h>	Jump functions

C Header Files	
<signal.h>	Signal handling functions
<stdarg.h>	Variable arguments handling functions
<stdio.h>	Standard Input/Output functions
<stdlib.h>	Standard Utility functions
<string.h>	String handling functions
<time.h>	Date time functions

- **Functions for files**:-

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

Function	description
fopen()	create a new file or open a existing file
fclose()	closes a file
getc()	reads a character from a file
putc()	writes a character to a file
fscanf()	reads a set of data from a file
fprintf()	writes a set of data to a file
getw()	reads a integer from a file

putw()	writes a integer to a file
fseek()	set the position to desire point
ftell()	gives current position in the file
rewind()	set the position to the begining point

- **Converting File Type:-**
  - Type conversion refers to changing a variable of one data type into another.
  - For instance, if you assign an integer value to a floating-point variable, the compiler will convert the int to a float.
  - Casting allows you to make this type conversion explicit, or to force it when it wouldn't normally happen.

➢ **Functions:-**

- **Designing:-**

Functions are an essential ingredient of all programs, large and small, and serve as our primary medium to express computational processes in a programming language. So far, we have discussed the formal properties of functions and how they are applied. We now turn to the topic of what makes a good function. Fundamentally, the qualities of good functions all reinforce the idea that functions are abstractions.

- Each function should have exactly one job. That job should be identifiable with a short name and characterizable in a single line of text. Functions that perform multiple jobs in sequence should be divided into multiple functions.
- *Don't repeat yourself* is a central tenet of software engineering. The so-called DRY principle states that multiple fragments of code should not describe redundant logic. Instead, that logic should be implemented once, given a name, and applied multiple times. If you find yourself copying and pasting a block of code, you have probably found an opportunity for functional abstraction.

- Functions should be defined generally. Squaring is not in the Python Library precisely because it is a special case of the pow function, which raises numbers to arbitrary powers.

These guidelines improve the readability of code, reduce the number of errors, and often minimize the total amount of code written. Decomposing a complex task into concise functions is a skill that takes experience to master. Fortunately, Python provides several features to support your efforts.

- **Structured programs:-**
  Structured programming is a programming paradigm aimed at improving the clarity, quality, and development time of a computer program by making extensive use of the structured control flow constructs of selection (if/then/else) and repetition (while and for), block structures, and subroutines.

- **Function in C:-**

A function is a block of statements that performs a specific task. Suppose you are building an application in C language and in one of your program, you need to perform a same task more than once. In such case you have two options –

a) Use the same set of statements every time you want to perform the task
b) Create a function to perform that task, and just call it every time you need to perform that task.

Using option (b) is a good practice and a good programmer always uses functions while writing codes in C.

Types of functions

1) **Predefined standard library functions** – such as puts(), gets(), printf(), scanf() etc – These are the functions which already have a definition in header files (.h files like stdio.h), so we just call them whenever there is a need to use them.

- 2) **User Defined functions –** The functions that we create in a program are known as user defined functions.

- **Inter-Function Communication:-**

When a function gets executed in the program, the execution control is transferred from calling a function to called function and executes function definition, and finally comes back to the calling function. In this process, both calling and called functions have to communicate with each other to exchange information. The process of exchanging information between calling and called functions is called inter-function communication.

In C, the inter function communication is classified as follows...

- **Downward Communication**
- **Upward Communication**
- **Bi-directional Communication**

- **Standard library functions:-**

The standard library functions are built-in functions in C programming.

These functions are defined in header files. For example,

The printf() is a standard library function to send formatted output to the screen (display output on the screen). This function is defined in the stdio.h header file.
Hence, to use the printf() function, we need to include the stdio.h header file using #include <stdio.h>.

The sqrt() function calculates the square root of a number. The function is defined in the math.h header file.

- **Passing Array to Functions**:-

Just like variables, array can also be passed to a function as an argument. In this guide, we will learn how to pass the array to a function using call by value and call by reference methods.

To understand this guide, you should have the knowledge of following C Programming topics:
C – Array

Function call by value in C
Function call by reference in C

Passing array to function using call by value method

As we already know in this type of function call, the actual parameter is copied to the formal parameters.

```
#include<stdio.h>
voiddisp(charch)
{
printf("%c ",ch);
}
int main()
{
chararr[]={'a','b','c','d','e','f','g','h','i','j'};
for(int x=0; x<10; x++)
{
/* I'm passing each element one by one using subscript*/
disp(arr[x]);
}

return0;
}
```

Output:

```
a b c d e f g h i j
```

Passing array to function using call by reference

When we pass the address of an array while calling a function then this is called function call by reference. When we pass an address as an argument, the function declaration should have a pointer as a parameter to receive the passed address.

```
#include<stdio.h>
voiddisp(int*num)
{
printf("%d ",*num);
}

int main()
{
intarr[]={1,2,3,4,5,6,7,8,9,0};
for(inti=0;i<10;i++)
{
/* Passing addresses of array elements*/
disp(&arr[i]);
}
```

```
return0;
}
```
Output:
1234567890

How to pass an entire array to a function as an argument?
In the above example, we have passed the address of each array element one by one using a for loop in C. However you can also pass an entire array to a function like this:
Note: The array name itself is the address of first element of that array. For example if array name is arr then you can say that arr is equivalent to the &arr[0].

```
#include<stdio.h>
voidmyfuncn(int*var1,int var2)
{
        /* The pointer var1 is pointing to the first element of
         * the array and the var2 is the size of the array. In the
         * loop we are incrementing pointer so that it points to
         * the next element of the array on each increment.
         *
         */
for(int x=0; x<var2; x++)
{
printf("Value of var_arr[%d] is: %d \n", x,*var1);
/*increment pointer for next element fetch*/
var1++;
}
}

int main()
{
intvar_arr[]={11,22,33,44,55,66,77};
myfuncn(var_arr,7);
return0;
}
```
Output:
Value of var_arr[0]is:11
Value of var_arr[1]is:22
Value of var_arr[2]is:33
Value of var_arr[3]is:44
Value of var_arr[4]is:55
Value of var_arr[5]is:66

Value of var_arr[6]is:77

- **Passing pointer to function:-**

In this example, we are passing a pointer to a function. When we pass a pointer as an argument instead of a variable then the address of the variable is passed instead of the value. So any change made by the function using the pointer is permanently made at the address of passed variable. This technique is known as call by reference in C.

```
#include<stdio.h>
voidsalaryhike(int *var, int b)
{
   *var = *var+b;
}
int main()
{
int salary=0, bonus=0;
printf("Enter the employee current salary:");
scanf("%d", &salary);
printf("Enter bonus:");
scanf("%d", &bonus);
salaryhike(&salary, bonus);
printf("Final salary: %d", salary);
return0;
}
```

Output:

```
Enter the employee current salary:10000
Enter bonus:2000
Final salary: 12000
```