

UNIT-I

Introduction to Computers, Creating and running Programs, Algorithm, Flow charts, Structure of C program.

Introduction to the C Language: Background, C Programs, Identifiers, Data Types, Variable, Constants, Input/output, Programming Examples.

Introduction to Computers:

What is a Computer?

The computer is an electronic device that operates under the control of instructions stored in its memory. A computer can take data from the user through input devices (**Input**), process the user-given data (**Processing**), produces the result to the user through output devices (**Output**), and stores data (**Information**) for future use. A Computer can be defined as follows:

The Computer is an electronic device that takes the data from the user as input, process that data, and produce the result.

A computer system is divided into two categories: Hardware and Software.

1. **Hardware** refers to the physical and visible components of the system such as a monitor, CPU, keyboard, and mouse.
2. **Software** refers to a set of instructions that enable the hardware to perform a specific set of tasks.

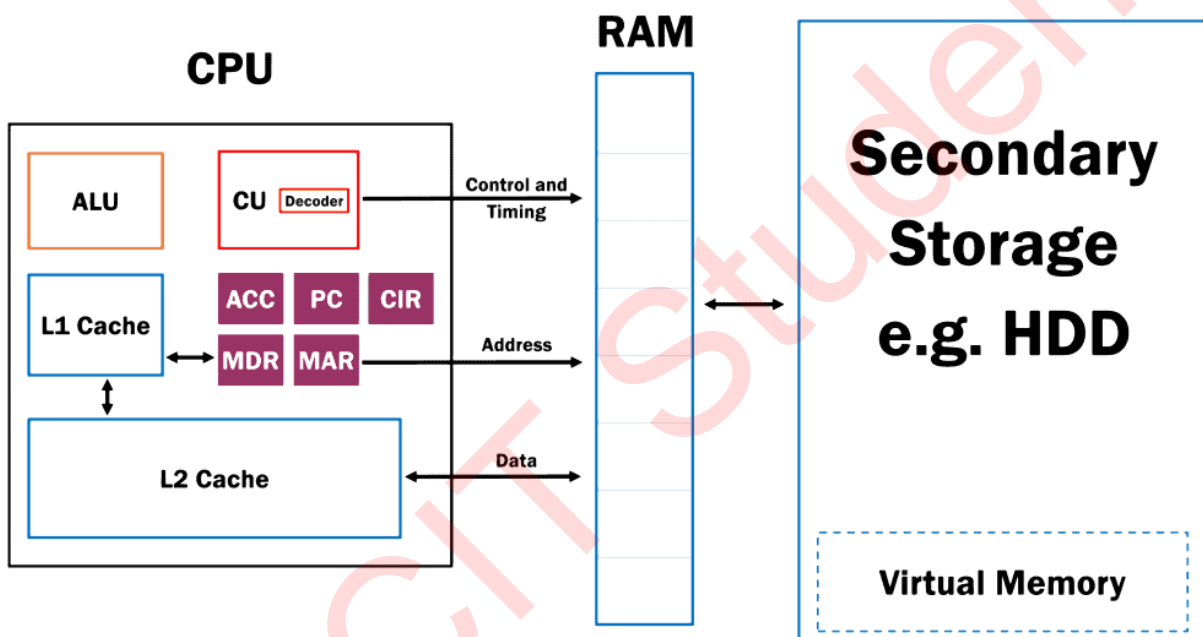


Computer Hardware

All physical components of the computer are called computer hardware. A user can see, touch, and feel every hardware of the computer. All hardware components perform any task based on the instructions given by the computer software.

Computer hardware is the physical part of a computer.

Computer Systems - Von Neumann Architecture



The computer hardware components are as follows.

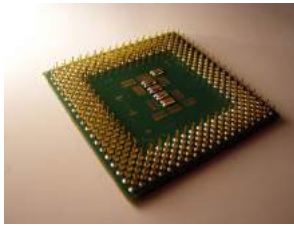
1. Input Devices - These are the parts through which a user can give the data to the computer.

Example: Keyboard, Mouse, Touch screen, Pen tablet, Speech MIC, Scanner

2. Output Devices - These are the physical components of a computer through which the computer gives the result to the user.

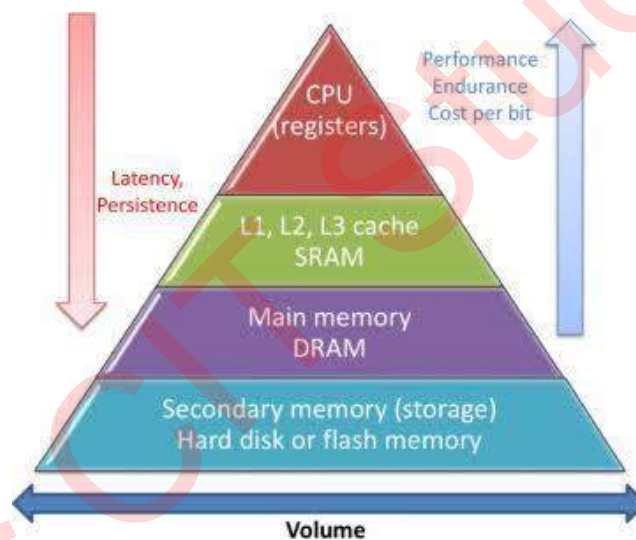
Example: Monitor, Speaker, Printer.

3. Processing unit - CPU





- **ALU** - Arithmetic Logic Unit
- **CU** - Control Unit
- **Registers** with L1& L2 Cache (CPU prioritizes memory lookup in this order: L1, L2, L3, and then in RAM)
 - PC(Program Controller), ACC (Accumulator Register),
 - IR (Instruction Register), CIR (Current Instruction Register),
 - MAR (Memory Address Register), MBR (Memory Buffer Register), MDR (Memory Data Register)

A software code such as C Program uses the CPU to read data in the memory units in the following order (Top to Bottom)






4. Storage Devices - These are the physical components of a computer in which the data can be stored and read. These are named Primary and Secondary storage devices

- **Primary Memory**

<p>ROM - Read Only Memory (BIOS, Read-Only Boot software)</p>	<p>Computer BIOS</p> 
<p>RAM - Random Access Memory (Temporary memory or Main memory. Cleared with reboot)</p>	<p>Crucial DDR DIMM</p> 

○ **Secondary Memory (Device Drives) -**

<p>CD</p>	
<p>HDD or SSD</p>	<p>SSD VS HDD</p>  <ul style="list-style-type: none"> - FASTER PERFORMANCE - NO VIBRATIONS OR NOISE - MORE ENERGY EFFICIENT - CHEAPER PER GB - AVAILABLE IN LARGE VERSIONS
<p>Pen/Flash drive</p>	

5. Devices Drives - Using drives, users can read and write data onto the storage devices like CDs, floppy, etc.

6. Cables - Various cables (Wires) are used to make connections in a computer.

7. Other Devices - Other than the above hardware components, a computer also contains components like a Motherboard, SMPS, Fans, etc.,

- **SMPS** - Switched Mode Power Supply (Power control & Voltage balance)



Software:

A set of computer programs are called software. A computer program is a series of instructions telling the computer what to do.

Types of software: **1. System software** **2. Application software**

1. System software:

System software exists in the functioning of a computer system and includes the operating system, assembler, interpreter, compiler, linker, and loader.

- **Operating System** is the interface between user applications and system hardware.
- **Assembler** is a translator that converses assembly language code into machine language.
- **Interpreter** converts source language program line by line into executable code at once.
- **Linker** performs the important task of linking together several object modules.
- The task of loading the linked object modules is performed by the **Loader**.

System software includes three types of programs:

- **Operating System:** The combination of a particular hardware configuration and system software package is known as a computer system platform. System platforms are commonly termed Operating Systems (OS). Some common operating systems are DOS, UNIX, Mac, and Windows platforms.
- **Language Translators:** These are interpreters and compilers for programs such as Pascal, BASIC, COBOL, C, and C++.
- **Common Utility Programs:** Communication tools, disk formatting tools, etc.

Examples: Language Translator, Operating System, Special Purpose Program, Utilities

2. Application software:

Application software is written to enable the computer to solve a specific data processing task. There are 2 categories of application software: **pre-written** software packages and **user application** programs.

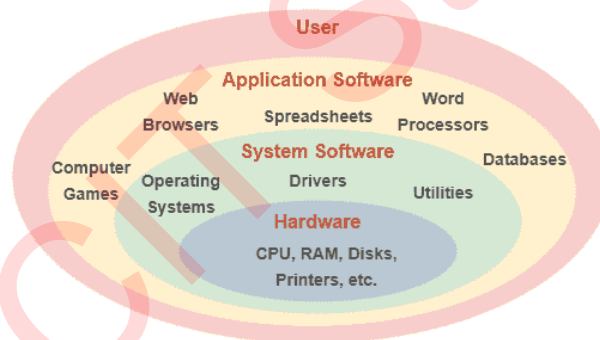
A number of powerful application software packages that do not require significant programming knowledge have been developed. These are easy to learn and use compared to programming languages.

Examples:

- Database Management Software (DBMS), ex: Oracle, MS-SQL, MySql
- Spreadsheet Software, ex: MS Spreadsheet / Google Sheets
- Word processing, Desktop Publishing (DTP), and Presentation Software, ex: MS Word / Google Docs, MS PPT or Google Slides
- Multimedia and Social Applications Software - Corporate email, Meeting application
- Data Communication Software
- Statistical and Operational Research Software
- Financial & Banking application, ex: Net banking, Bank apps, PayTm, GPay

Summary of Hardware & Software:

OS (Operating System) is a System software that acts as an interface between computer hardware and the user. The OS controls the essential management functions on memory, process, files, and input/output. This interface helps users interact with hardware devices and give instructions to the system. Users can install a specific application software to perform a particular task such as browsing, image editing, opening media files, etc.



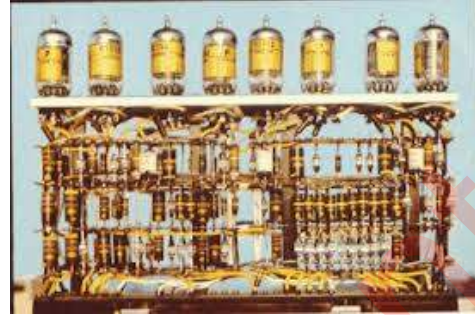
Generation of computer:

Generation means variation between different hardware technologies. The generations of computers are mainly classified from 1st generation to 5th generation as of now.

1st generation: (1946-1956)

First-generation computers are made with the use of **Vacuum Tubes**. These computers used machine language for programming.

Example: ENIAC(Electronic numeric integrator and computer)UNIVAC(Universal Accounting company)



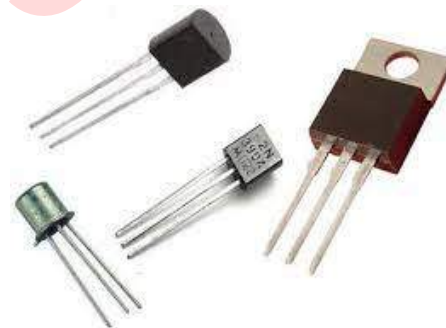
Disadvantages:

1. Occupied lot of space
2. Consumed a lot of power
3. Produced a lot of heat
4. Costly system

2nd generation: (1957-1963)

The computer in which Vacuum Tubes were replaced by **Transistors** are called the second generation of computers these computers used assembly language for programming.

Example: IBM 7090, IBM 7094



Advantages:

1. Less expensive
2. Consumed less power
3. Produced little heat
4. Less cost and work at higher speed

3rd generation: (1964-1981)

The computers using **Integrated Circuits(ICs)** came to be known as the third generation of computers. These use high-level language.

Example: IBM 370, Cyber 175



4th generation: (1982-1989)

The computers which were built with **Microprocessors** are identified as the fourth generation computers. These computers use VLSI (Very Large Scale Integration) microchips for both CPU and memory. **VLSI** microprocessors contain thousands of electronic components in a single IC.

Example: CRAY-2, IBM 3090



5th generation: (1990- till now)

Fifth generation computers are under the development stage these computers use ULSI (Ultra Large Scale Integration) microprocessors. ULSI microchips contain millions of electronic components such as transistors in a single IC. The aim of these computers is to develop Artificial Intelligence.



Features and Advantages

- Development of AI (Artificial Intelligence) and NLP (Natural Language Processing)
- Advancement in Parallel Processing and Superconductor technology
- Better user experience with multimedia features
- Availability of smaller yet powerful computers at a cheaper cost

Bits and Bytes:

Bits:

- The term bit is an acronym of **B**inary **D**igit.
- The smallest unit of storage is called a bit.
- A bit is the smallest unit of data in a computer.
- A bit has a single binary value, either 0 or 1.
- 8 bits = 1 byte.
- A single byte(8 bits) can store the values 0-255 or one letter.

Byte:

A pattern of eight bits makes a BYTE. A BYTE represents characters, letters, numbers, and symbols

8 bits are 1 **Byte**

16 bits are 2 **Bytes** = 1 **halfword**

32 bits are 4 **Bytes** = 1 **word**

1024 Bytes are 1 **Kilobyte (KB)**

1024 Kilobytes are 1**Megabyte (MB)**

1024 Megabytes are 1**Gigabyte (GB)**

1024 Gigabytes are 1**Terabyte (TB)**

1024 Terabytes are 1 **Petabyte (PB)**

1024 Petabytes are 1 **Exabyte (EB)**

1024 Exabytes are 1 **Zettabyte (ZB)**

1024 Zettabytes are 1 **Yottabyte (YB)**

Creating and Running C Program:

To create and execute C programs in the 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 a Source Code

Source code is a file with C programming instructions in a 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 a source code file in Windows OS.

Click on the **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 Editor window

Type the **program**

Save it as **FileName.c** (Use shortcut key **F2** to save)

Step 2: Compile Source Code (Alt + F9)

The 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**.

The compilation is the process of converting high-level language instructions into low-level language instructions.

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 Errors. If there are any Errors then the compiler returns a List of Errors, if there are no errors then the source code is converted into **object code** and stores it as a 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 a **.exe** extension.

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

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

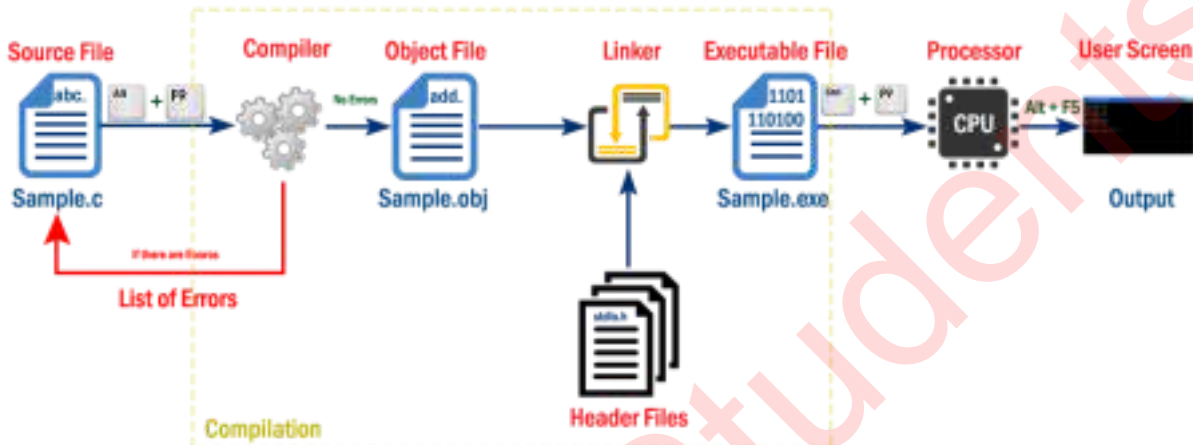
We use a shortcut key **Ctrl + F9** to run a C program. Whenever we press **Ctrl + F9**, the **.exe** file is submitted to the **CPU**. On receiving **.exe** file, the **CPU** performs the task according to the instruction written in the file. The result generated from the execution is placed in a window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

**What is an algorithm?**

An algorithm is a step-by-step procedure to solve a problem. In normal language, the algorithm is defined as a sequence of statements that are used to perform a task. In computer science, an algorithm can be defined as follows.

An algorithm is a sequence of unambiguous instructions used for solving a problem, which can be implemented (as a program) on a computer.

Specifications of Algorithms

Every algorithm must satisfy the following specifications...

1. **Input** - Every algorithm must take zero or more number of input values from external.
2. **Output** - Every algorithm must produce an output as result.
3. **Definiteness** - Every statement/instruction in an algorithm must be clear and unambiguous (only one interpretation).
4. **Finiteness** - For all different cases, the algorithm must produce result within a finite number of steps.
5. **Effectiveness** - Every instruction must be basic enough to be carried out and it also must be feasible.

Examples:

Algorithm 1: Add two numbers entered by the user

Step 1: Start

Step 2: Declare variables num1, num2, and sum.

Step 3: Read values num1 and num2.

Step 4: Add num1 and num2 and assign the result to sum. $sum \leftarrow num1 + num2$

Step 5: Display sum

Step 6: Stop

Algorithm 2: Find the largest number among three numbers Step 1: Start

Step 2: Declare variables a,b, and c.

Step 3: Read variables a,b and c.

Step 4:

If $a > b$

 If $a > c$

 Display a is the largest number.

 Else

 Display c is the largest number.

Else If $b > c$

 Display b is the largest number.

 Else

 Display c is the greatest number.

Step 5: Stop

Algorithm 3: Find the factorial of a number

Step 1: Start

Step 2: Declare variables n, factorial and i.

Step 3: Initialize variables

 factorial \leftarrow 1

 i \leftarrow 1

Step 4: Read value of n

Step 5: Repeat the steps until $i = n$

5.1: factorial \leftarrow factorial*i

5.2: i \leftarrow i+1

Step 6: Display factorial

Step 7: Stop

Algorithm 4: Check whether a number is prime or not Step 1: Start

Step 2: Declare variables n, i, flag.

Step 3: Initialize variables

flag \leftarrow 1

i \leftarrow 2

Step 4: Read n from the user.

Step 5: Repeat the steps until i=(n/2)

5.1 If remainder of n÷i equals 0

flag \leftarrow 0

Go to step 6

5.2 i \leftarrow i+1

Step 6: If flag = 0

Display n is not prime







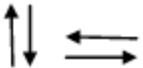
else

Display n is prime

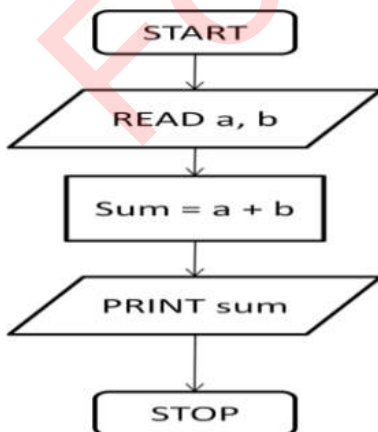
Step 7: Stop

Flowchart:

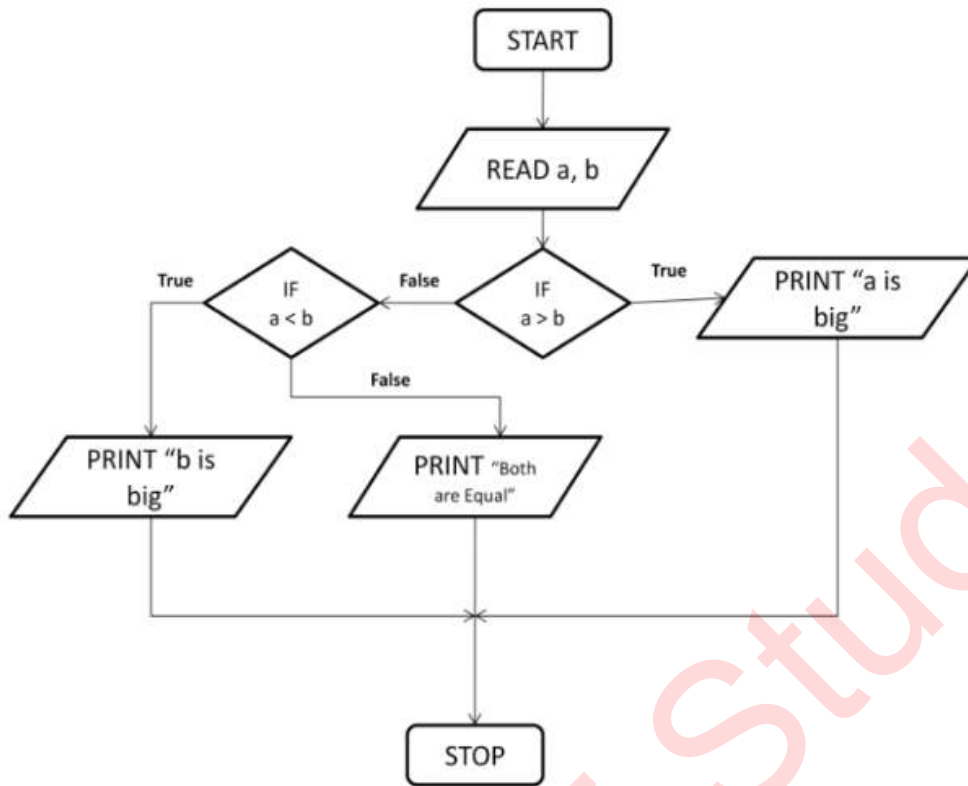
A flowchart is a pictorial representation of an algorithm. It shows the logic of the algorithm and the flow of control. The flowchart uses symbols to represent specific actions and arrows to indicate control flow. The Flowchart symbols are Flow of control

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

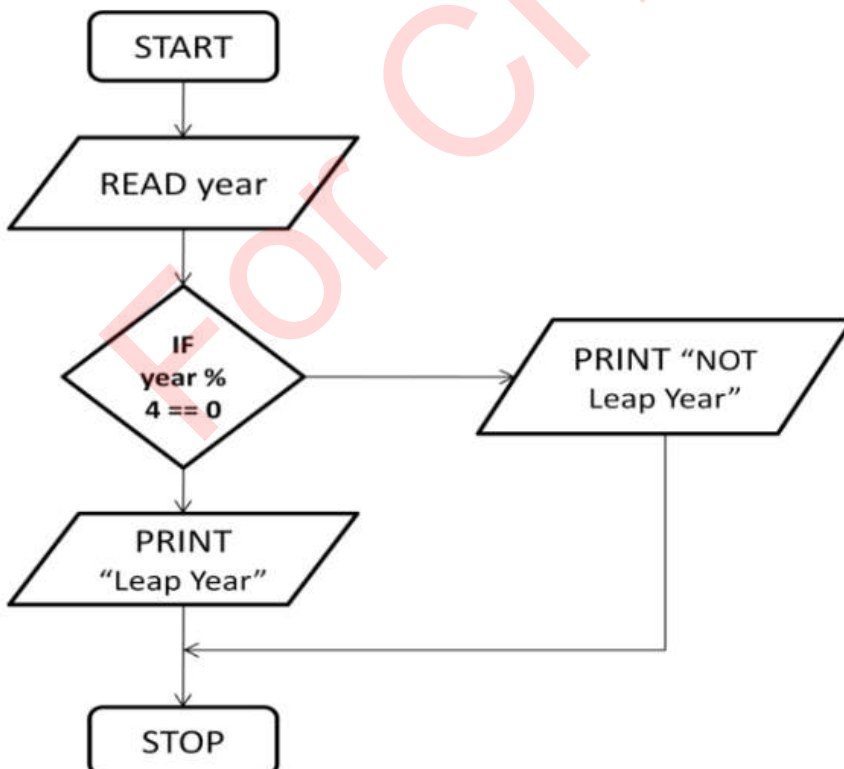
1. Example Flow chart for Addition of two numbers is:



2. Example Flowchart for Biggest of two numbers:



3. Example Flowchart for finding Leap Year:



Basic Structure of C Program:

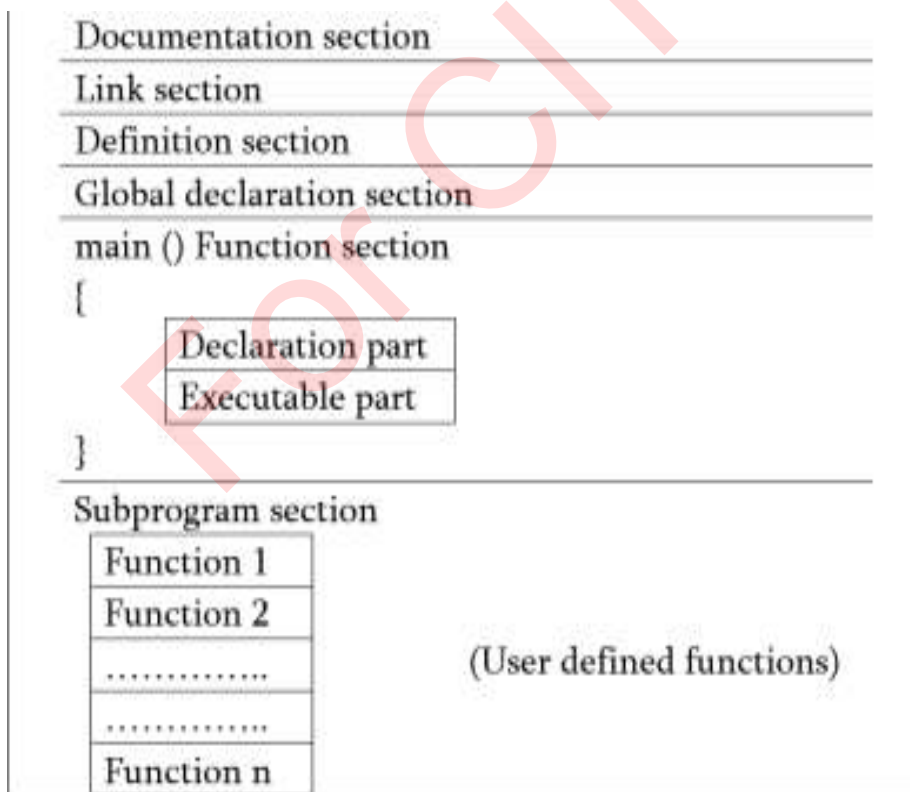
Almost every programming language has a structure. The C language also has a structure.

A C program is divided into six sections:

1. Documentation
2. Linking (preprocessor directive)
3. Definition (preprocessor directive)
4. Global Declaration
5. Main() Function
 - a. Local Declaration part
 - b. Execution part
6. Subprograms
 - a. Local Declaration part
 - b. Execution part

Each of these sections has a purpose. These sections make the program easy to read, easy to modify, easy to document, and makes it consistent in format.

The execution of a C program starts from main() function. Hence, the main() section is compulsory. The rest of the sections are optional in the structure of a C program.



Program Section	Description
Documentation	Generally contains the program description, author's name, date of creation, and version number. No syntax to follow; the C compiler ignores this section.
Link	Needed header files are included in this section which contains various functions from the C library. A copy of these header files is inserted into your program code before compilation.
Definition	Includes preprocessor directives that contain symbolic constants or macros. Ex: #define helps to use constants in our code. It replaces all the constants with its value in the code.
Global Declaration	Includes declaration of global variables, function declarations, static global variables, and functions. There are some variables that are used in more than one function. Such variables are called global variables and are declared in the global declaration section that is outside of all the functions. This section also declares all the <i>user-defined functions</i> .
Main() Function	<p>For every C program, the execution starts from the main() function. It is mandatory to include a main() function in every C program to run it.</p> <p>Every C program must have one main function section. This section contains two parts; the declaration part and the executable part</p> <ol style="list-style-type: none"> 1. Declaration part: The declaration part declares all the <i>variables</i> used in the executable part. 2. Executable part: There is at least one statement in the executable part. These two parts must appear between the opening and closing braces. The <i>program execution</i> begins at the opening brace and ends at the closing brace. The closing brace of the main function is the logical end of the program. All statements in the declaration and executable part end with a semicolon.
Subfunctions	<p>Include user-defined functions (functions the programmer writes). These functions contain the inbuilt C functions and the function definitions that are already declared in the Global Declaration section. These functions are called from the main() function.</p> <p>If the program is a <i>multi-function program</i> then the subprogram section contains all the <i>user-defined functions</i> that are called in the main () function. User-defined functions are generally placed immediately after the main () function, although they may appear in any order.</p>

Note: A well-structured C program makes debugging (or corrections) easier and increases the readability and modularity (modules or parts) of the code.

A Sample to Demonstrate 6 Sections in the Structure of C Program:

```

//Section 1: Documentation area
/* Description: Program to find age and result
 * File: ageresult.c
 * Author: your name
 * Version: 1.0
 */

//Section 2: Linking of C header files or user library files
#include<stdio.h>      //Link to a header file in C library
#include<conio.h>     //Link to a header file in C library
#include<math.h>      //Link to a header file in C library

//Section 3: Definition area for Preprocessor directives & Constants
#define BIRTH 2004    //Definition of a macro
#define PASS 25      //Definition of a macro

//Section 4: Global Declaration of Functions and Variables
int age(int current); //Global Declaration of function
char grade(int marks); //Global Declaration of function
int marks_a=70;      //Global Declaration of variable

//Section 5: main() function Definition area
int main()           //Main() Function
{ int year,tmarks;
  printf("\n Enter current year: ");
  scanf("%d",&year);
  printf(" Your Age: %d", age(year)); //function call from main()

  printf("\n Enter Total Marks: ");
  scanf("%d",&tmarks);
  printf(" Your Grade: %c", grade(tmarks)); //function call from main()

  return 0;
}

//Section 6: Subfunction Definition area
//(Or, Definitions of user defined functions)
int age(int a) //subfunction definition
{
    return (a-BIRTH); //sends result back to main()
}

```

```
char grade(int marks) //subfunction definition
{
    char res;
    res = (marks>=PASS)?'P':'F'; //Ternary operator.
    return res; //sends the result back to main()
}
```

Output:

Enter current year: 2022

Your Age: 18

Enter Total Marks: 50

Your Grade: P

A Simple Program to the Find Area of Circle in C Language

```
/* Description: A Program to find Area of Circle In C Language
   Author Name: name
   Date: 27/10/2022   */
#include<stdio.h> // Link Section
#define PI 3.14 // Definition Section
float r; // Global Declarations Section
float areaofcircle(float); // Global Declarations Section
int main(void) // Main functions Section
{
    float aoc; // Declaration Part
    printf("Enter the radius of circle: "); // Execution Part
    scanf("%f",&r);
    aoc=areaofcircle(r);
    printf("Area of circle is %f",aoc);
    return (0);
}

float areaofcircle(float R) // User-Defined Functions or Sub Program Section
{
    return (R*R*PI) ;
}
```

Output -:

Enter the radius of circle: 5

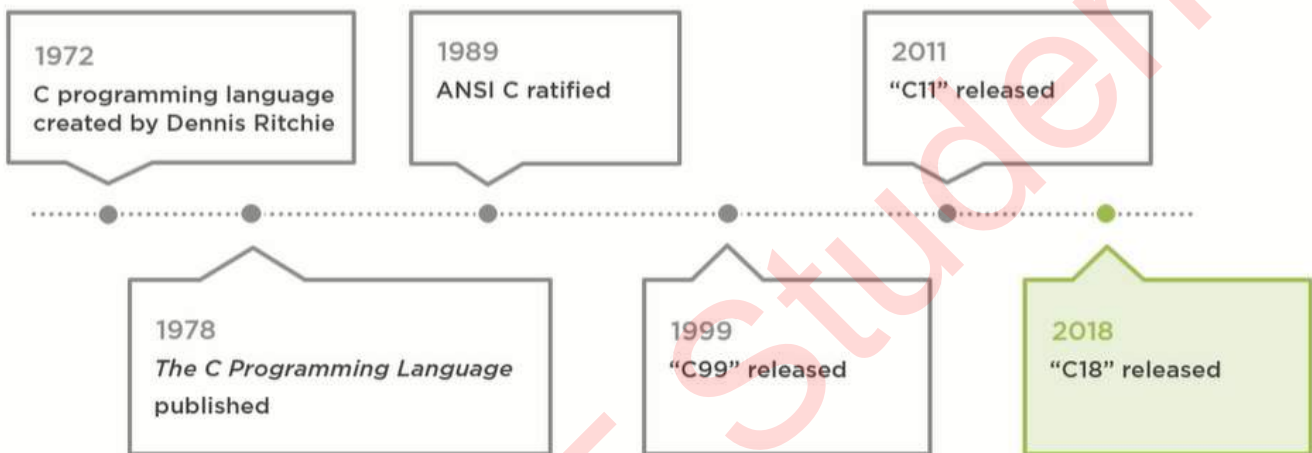
Area of circle is 78.500000

Introduction to C Programming

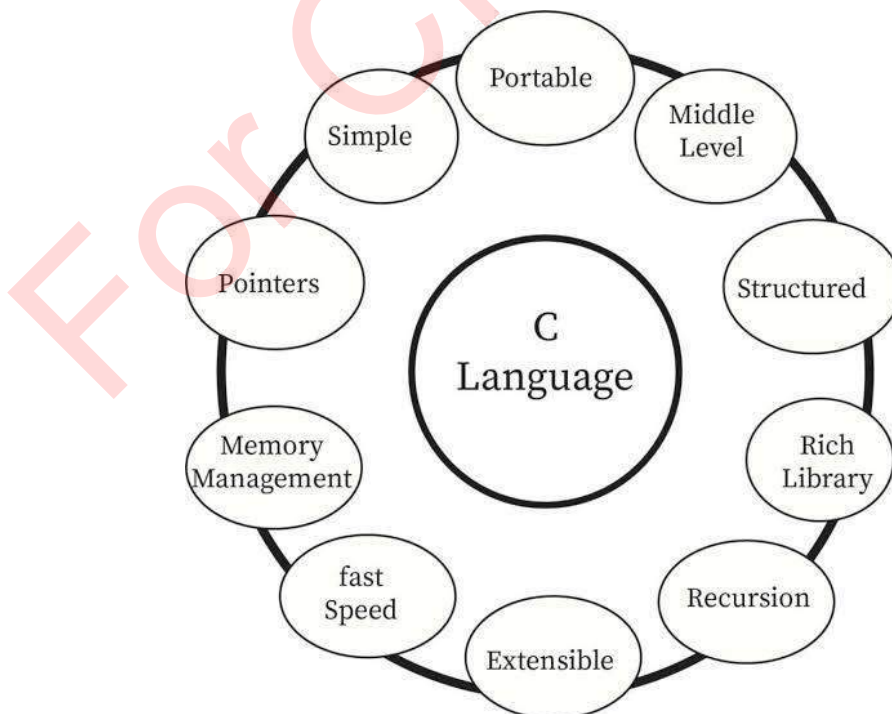
C Background

C programming language was developed in 1972 by Dennis Ritchie at bell laboratories of AT&T (American Telephone & Telegraph), located in the U.S.A. **Dennis Ritchie** is known as the **founder of the c language**.

It was developed to overcome the problems of previous languages such as B, BCPL, etc. Initially, C language was developed to be used in **UNIX operating system**. It inherits many features of previous languages such as B and BCPL. The evolution of C Language to date is shown below.



Features of C Language



C is the widely used language. It provides many **features** that are given below.

1. Simple
2. Machine Independent or Portable
3. Mid-level programming language
4. Structured programming language
5. Rich Library
6. Memory Management
7. Fast Speed
8. Pointers
9. Recursion
10. Extensible

1) Simple: C is a simple language in the sense that it provides a structured approach (to break the problem into parts), a rich set of library functions, data types, etc.

2) Machine Independent or Portable: Unlike assembly language, c programs **can be executed on different machines** with machine-specific changes. Therefore, C is a machine-independent language.

3) Mid-level programming language: C has the features of both assembly-level languages i.e low-level languages and higher-level languages. So the C is generally called a middle-level Language. The user uses C language for writing an operating system (low level) and also generates menu driven customer billing system (high level).

4) Structured programming language: C is a structured programming language in the sense that we can break the program into using functions. So, it is easy to understand and modify. Functions also provide code reusability which means define once and use many times.

5) Rich Library: C provides a lot of inbuilt functions that make development fast.

6) Memory Management: It supports the feature of dynamic memory allocation. In C language, we can free all memory at any time by calling the free() function.

7) Speed: The compilation and execution time of C language is fast since there are lesser inbuilt fun and hence lesser overhead.

8) Pointer: C provides the feature of pointers. We can directly interact with the memory by using pointers. We can use pointers for memory, structures, functions, array, etc.

9) Recursion: In C, we can call the function within the function. It provides code reusability for function. Recursion enables us to use the approach of backtracking.

10) Extensible: C language is extensible because it can easily adopt new features

C Character Set:

As every language contains a set of characters used to construct words, statements, etc., C language also has a set of characters which include **alphabets, digits, and special symbols**. C language supports a total of 256 characters.

Every C program contains statements. These statements are constructed using words that are constructed using characters from the C character set that contains the following set of characters.

- 1. Alphabets 2. Digits 3. Special Symbols**

Alphabets

C language supports all the alphabets from the English language. Lower and upper case letters together support 52 alphabets.

- lower case letters - **a to z** UPPER CASE LETTERS - **A to Z**

Digits

C language supports 10 digits which are used to construct numerical values in C language. Digits - **0, 1, 2, 3, 4, 5, 6, 7, 8, 9**

Special Symbols

C language supports a rich set of special symbols that include symbols to perform mathematical operations, to check conditions, white spaces, backspaces, and other special symbols.

Special Symbols - ~ @ # \$ % ^ & * () _ - + = { } [] ; : ' " / ? . > , < \ | **tab** newline space NULL bell backspace vertical tab, etc.,

Every character in C language has its equivalent ASCII (American Standard Code for Information Interchange) value.

C Tokens

Every C program is a collection of instructions and every instruction is a collection of some individual units. Every smallest individual unit of a c program is called a token. Every instruction in a c program is a collection of tokens. Tokens are used to construct c programs and they are said to be the basic building blocks of a c program. **Tokens in a C program:**

<ol style="list-style-type: none"> 1. Keywords 2. Identifiers 3. Operators 4. Special Symbols 5. Constants 6. Strings 7. Data values 	<p>In a C program, a collection of all the keywords, identifiers, operators, special symbols, constants, strings, and data values are called tokens.</p>
---	---

C Keywords

As every language has words to construct statements, C programming also has words with a specific meaning which are used to construct c program instructions. In the C programming language, keywords are special words with predefined meanings. Keywords are also known as reserved words in the C programming language.

In the C programming language, there are **32 keywords**. All 32 keywords have their meaning which is already known to the compiler.

Keywords are reserved words with predefined meanings which already known to the compiler

Whenever the C compiler come across a keyword, automatically it understands its meaning.

Properties of Keywords

1. All the keywords in C programming language are defined as lowercase letters so they must be used only in lowercase letters
2. Every keyword has a specific meaning; users can not change that meaning.
3. Keywords cannot be used as user-defined names like variables, functions, arrays, pointers, etc...
4. Every keyword in C programming language represents something or specifies some kind of action to be performed by the compiler.

The following table specifies all 32 keywords.

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

C Identifiers

In the C programming language, programmers can specify their name to a variable, array, pointer, function, etc. An identifier is a collection of characters that acts as the name of a variable, function, array, pointer, structure, etc. In other words, an identifier can be defined as the user-defined name to identify an entity uniquely in the c programming language. That name may be of the variable name, function name, array name, pointer name, structure name, or a label.

The identifier is a user-defined name of an entity to identify it uniquely during the program execution

Example

```
int marks;  
char studentName[30];
```

Here, **marks** and **studentName** are identifiers.

Rules for Creating Identifiers

1. An identifier can contain **letters** (UPPERCASE and lowercase), **numerics** & **underscore** symbols only.
2. An identifier should not start with a numerical value. It can start with a letter or an underscore.
3. We should not use any special symbols in between the identifier even whitespace. However, the only underscore symbol is allowed.
4. Keywords should not be used as identifiers.
5. There is no limit to the length of an identifier. However, the compiler considers the first 31 characters only.
6. An identifier must be unique in its scope.

Valid Identifiers:

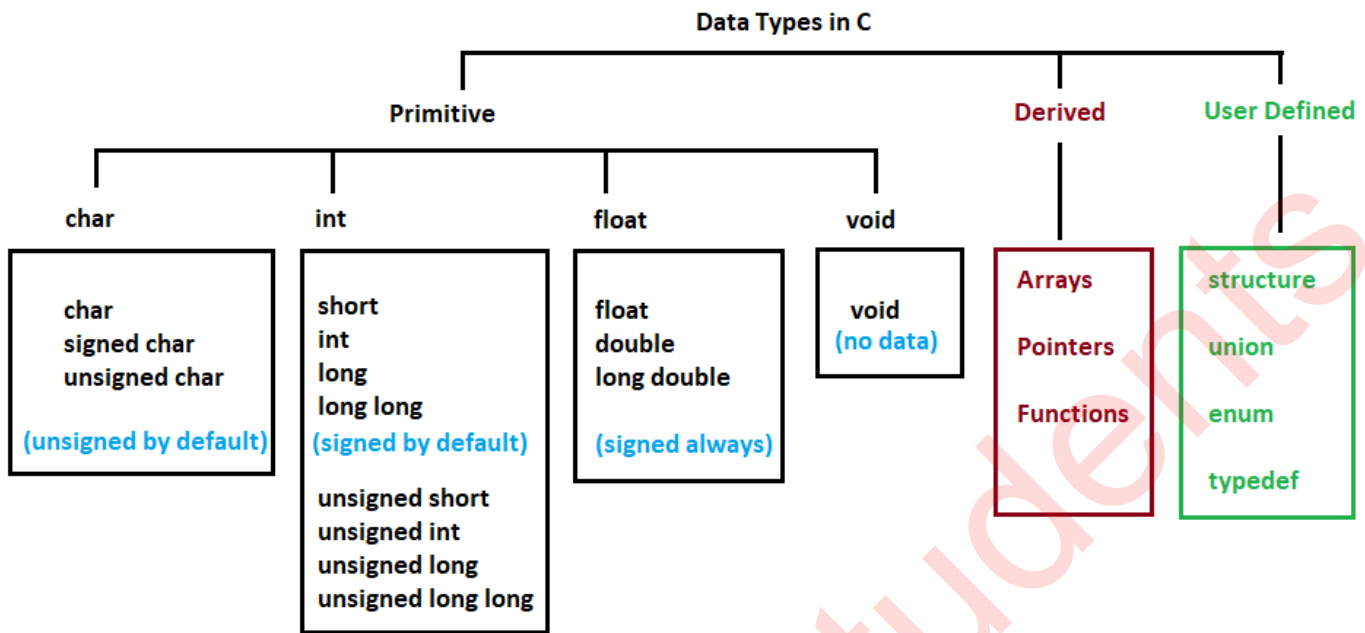
- bonus (It contains only lowercase alphabets)
- total_sum (It contains only '_' as a special character)
- _salary (It starts with an underscore '_')
- area_ (Contains lowercase alphabets and an underscore)
- num1 (Here, the numeric digit comes at the end)
- num_2 (It starts with lowercase and ends with a digit)

Invalid Identifiers:

- 5salary (it begins with a digit)
- \@width (starts with a special character other than '_')
- int (it is a keyword)
- m n (contains a blank space)
- m+n (contains a special character)

Data Types in C

A data type specifies the type of data that a variable can store such as integer, floating, character, etc.



a) Primitive / Basic Data Type:

Integer data types:

C offers 4 different integer data types: **int**, **short int**, **long int**, **long long int**. The difference between these 4 integer types are bytes required and the range of values. Formulas to calculate the range for,

signed data type: -2^{n-1} to $2^{n-1}-1$ where n = size in the number of bits

unsigned data type: 0 to 2^n-1 where n = size in the number of bits

TYPE	SIZE (bytes)	SIZE (bits)	RANGE
short int	2	16	-2^{15} to $2^{15}-1$
int	2 or 4	32	-2^{31} to $2^{31}-1$
long int	4	32	-2^{31} to $2^{31}-1$
long long int	8	64	-2^{63} to $2^{63}-1$
unsigned short int	2	16	0 to $2^{16}-1$
unsigned int	2 or 4	32	0 to $2^{32}-1$
unsigned long int	4	32	0 to $2^{32}-1$
unsigned long long int	8	64	0 to $2^{64}-1$

PTC UNIT-1

Float data types:

Like integer, float are divided into 3 different individual data types they are float, double, long double. The differences between these 3 individual types are bytes required and range of values.

TYPE	SIZE (bytes)	SIZE (bits)	RANGE
float	4	32	3.4E-38 to 3.4E+38
double	8	64	1.7E-308 to 1.7E+308
long double	10 (12 or 16 as per C99/C11)	80	3.4E-4932 to 3.4E+4932

Character data types:

Keyword **char** is used for declaring character type variables.

Example: char a= 'm';

TYPE	SIZE (bytes)	SIZE (bits)	RANGE
char	1	8	-128 to 127
unsigned char	1	8	0 to 255

Data Format Specifiers in C

In C programming language, values can be type integer, floating-point, single character, or sequence of characters. Format specifiers tell the compiler about the data type of the value we give as input or print as output.

C contains different format specifiers used in printf() and scanf() functions. The most commonly used format specifiers in C programs are:

Data Type	Format Specifier	Used purpose	Size (bytes)	Size (bits)	Range
char	%c	single character	1	8	-128 to 127
signed char	%c	single character	1	8	0 to 255
char	%s	string of chars	1	8	
short int	%hd	short (signed) integer	2	16	-32,768 to 32,767 (-2 ¹⁵) to (2 ¹⁵ - 1)
unsigned short int	%hu	short (unsigned) integer	2	16	0 to 65,535 0 to (2 ¹⁶ -1)

PTC UNIT-1

int	%d	integer (assumes base 10)	2 or 4	16 or 32	(-2^{31}) to $(2^{31} - 1)$ -2,147,483,648 to 2,147,483,647
int	%i	integer (detects the base automatically)	2 or 4	16 or 32	(-2^{31}) to $(2^{31} - 1)$ -2,147,483,648 to 2,147,483,647
unsigned int	%u	int unsigned	2 or 4	16 or 32	0 to $(2^{32}-1)$ 0 to 4,294,967,295
long int	%ld	long integer	4	32	(-2^{31}) to $(2^{31} - 1)$ -2,147,483,648 to 2,147,483,647
unsigned long int	%lu	unsigned long integer	4	32	0 to $(2^{32}-1)$ 0 to 4,294,967,295
long long int	%lld	long long integer	8	64	(-2^{63}) to $(2^{63} - 1)$
unsigned long long int	%llu	unsigned long long integer	8	64	0 to $(2^{64}-1)$
float	%f	floating point number for floats	4	32	
double	%lf	double (large float)	8	64	
long double	%Lf	long double (larger float)	10 (12 / 16 in C99/C11)	80	
float / double / long double	%e	floating point number in scientific notation			
float / double / long double	%E	floating point number in scientific notation			
Ex: int	%o %#o	an octal (base 8) integer			(%#o prepends 0 to oct num)
Ex: int	%x %#x	a hexadecimal (base 16) integer			(%#x prepends 0x to hex num)
any data type	%p	an address (or pointer)			
	%n	prints nothing			
	%%	the % symbol			

Example: c2_format_specifiers1.c

```
//Format specifiers in C
#include <stdio.h>
int main()
{   //%c Character
    char code;
    printf("\nEnter a character : ");
    scanf("%c",&code);
    printf("\nThe character is: %c \n",code);

    //%s String
    char name[15]="Programming";
    printf("The string value of s is %s \n",name);

    //%d Decimal Integer base 10, %i Integer detects base
    short int rank=50;
    printf("The integer value of a is %d \n",rank);
    printf("The integer value of a is %i \n",rank);

    //%f Floating Point
    float price=12.5;
    printf("The floating point of price is %f \n",price);
    //%e Floating Pointer Number
    printf("The floating-point of price is %e \n",price);
    printf("The floating-point of price is %E \n",price);

    //%lf Double
    double d=127.5;
    printf("The double value of d is %lf \n",d);

    //%o octal integer
    int p=11;
    printf("The octal integer value of %d is %o \n",p,p);

    //%x Hexadecimal Integer
    int q=14;
    printf("The hexadecimal value of %d is %x \n",q,q);

    //%p Prints Memory Address
    int sum=0;
    printf("The memory address of sum is %p \n",&sum);
    return 0;
}
```

Output:

Enter a character : A

The character is: A

The string value of s is Programming

The integer value of a is 50

The integer value of a is 50

The floating point of price is 12.500000

The floating-point of price is 1.250000e+001

The floating-point of price is 1.250000E+001

The double value of d is 127.500000

The octal integer value of 11 is 13

The hexadecimal value of 14 is e

The memory address of sum is 0061FEF4

Example: c2_format_specifiers2.c

```

/* printf() - Format specifiers */
#include <stdio.h>
int main() {
    printf ("Integers: %i %u \n", -3456, 3456);
    printf ("Characters: %c %c \n", 'z', 80);
    printf ("Decimals: %d %ld\n", 1997, 32000L);
    // %#x prepends 0x to hex num and %#o prepends 0 to oct num
    printf ("Same number in Different formats: %d %x %o %#o %#x\n",
           100, 100, 100, 100, 100);
    printf ("Different float formats: %4.2f %+.0e %E \n",
           3.14159, 3.14159, 3.14159);
    printf ("Preceding with empty spaces: %10d \n", 1997);
    printf ("Preceding with zeros: %010d \n", 1997);
    printf ("%s \n", "Programming");
    return 0;
}

```

Output:

Integers: -3456 3456

Characters: z P

Decimals: 1997 32000

Same number in Different formats: 100 64 144 0144 0x64

Different float formats: 3.14 +3e+000 3.141590E+000

Preceding with empty spaces: 1997

Preceding with zeros: 0000001997

Programming

b) Derived Data Types:

Derived data types are those that are defined using the primitive of base data types. Derived data types are those whose variables allow us to store multiple values of the same type. But they never allow storing multiple values of different types. **Arrays, Pointers, and Functions** are derived data types. We will discuss these data types in detail during later units.

c) User-Defined Data Types:

The user-defined datatypes are two types. They are

1. Structure (**struct**)
2. Union (**union**)
3. Type definition (**typedef**)
4. Enumerated datatype (**enum**)

1. Structure

A structure is an user-defined data type. It is a collection of different data types in a single entity called a structure. The elements of the structure are known as its members. The members can be accessed by using the dot (.) operator.

```
struct emp
{
    int id;
    char name[30];
    int sal;
}

struct emp e1, e2;
e2.id = 102;
strcpy (e2.name, "Rajesh")
```

2. Union

A union is an user-defined data type and is a collection of different data types in a single entity called a union. The elements of the structure are known as its members. The members can be accessed by using the dot (.) operator.

union Cars

```
{ int numberOfCars;
  float price;
  char brand[20];
};

union Cars car;
car.numberOfCars = 10;
```

3. Type definition:

The users can create an additional name (alias) for another data type, but it does not create a new type

Ex: `typedef long int lno;`

```
lno n1,n2;
```

4. Enumerated datatype:

It is a user-defined data type in C. It is mainly used **to assign names to integer constants**. These names make a program easy to read and maintain. After that, the user can declare variables using this new enum type.

```
#include<stdio.h>
enum weekday{sun, mon, tue, wed, thu, fri, sat};
int main (){
    enum weekday day;
    day = wed;
    printf("Day %d", day);
    return 0;
}
```

Output: 3 (default integer values for enum names are 0,1,2,3...)

Variables in C

A **variable** is a name of the memory location. It is used to store data. Its value can be changed, and it can be reused many times.

It is a way to represent memory location through symbols so that it can be easily identified.

Let's see the **syntax** to declare a variable: `type variable_list;`

The examples of declaring variables of different data types:

1. `int a;` //declaring variable a of integer type
2. `float b;` //declaring variable b of float type
3. `char c;` //declaring variable c of char type

Here, a, b, c are variables. The int, float, and char are the data types. We can also assign values while declaring the variables with a data type; assigning a value to a variable is called initialization.

1. `int a=10, b=20;` //declaring & initializing variables a and b of integer type
2. `float f=20.8;` //declaring & initializing variable f of float type
3. `char c='A';` //declaring & initializing variable c of char type

Rules for defining variables

- A variable can have alphabets, digits, and underscore.
- A variable name can start with the alphabet, and underscore only. It can't start with a digit.
- No whitespace is allowed within the variable name.
- A variable name must not be any reserved word or keyword, e.g. int, float, etc.

Valid variable names:

1. `int a;`
2. `int _ab;`
3. `int a30;`

Invalid variable names:

1. `int 2;`
2. `int a b;`
3. `int long;`

Types of Variables in C

There are many types of variables in c:

1. local variable
2. global variable
3. static variable
4. automatic variable
5. external variable
6. register variable

Local Variable:

A variable that is declared inside the function or block is called a local variable. It must be declared at the start of the block.

```
void function1(){  
    int x=10; //local variable  
}
```

You must have to initialize the local variable before it is used.

Global Variable:

A variable that is declared outside the function or block is called a global variable. Any function can change the value of the global variable. It is available for all functions.

It must be declared at the start of the block.

```
int value = 20; //global variable
void main() {
    int x=10; //local variable
}
```

Static Variable:

A variable that is declared with the static keyword is called a static variable. It retains its value between multiple function calls.

```
void function1() {
    int x=10; //local variable
    static int y=10; //static variable
    x=x+1;
    y=y+1;
    printf("%d,%d",x,y);
}
```

If you call this function many times, the **local variable will print the same value** for each function call, e.g. 11,11,11 and so on. But the **static variable will print the incremented value** in each function call, e.g. 11, 12, 13, and so on.

Automatic Variable:

All variables in C that are declared inside the block, are automatic variables by default. We can explicitly declare an automatic variable using the **auto keyword**.

```
void main() {
    int x=10; //local variable (also automatic)
    auto int y=20; //automatic variable
}
```

External Variable:

We can share a variable in multiple C source files by using an external variable. To declare an external variable, you need to use the **extern** keyword.

myfile.h

```
int x=10;
```

program1.c

```
#include "myfile.h"
#include <stdio.h>
extern int x; //external variable (also global)
void printValue() {
    printf("Global variable: %d", global_variable);
}
```

Register Variable:

Register variables **tell the compiler to store the variable in CPU register instead of RAM memory.**

- Frequently used variables with small values are kept in CPU Registers (small caches of L1, L2, L3) and these variables can be accessed faster than the variables saved in the RAM.
- We cannot get the addresses of these variables. “**register**” keyword is used to declare the register variables

Example:

register int count = 10; //variables for loop counters can be declared as register.

```
#include <stdio.h>
int main() {
    register int count = 10;
    auto int a = 7;
    printf("\n Value of register variable count : %d",count);
    printf("\n Value of auto variable : %d",a);
    printf("\n Sum of register & auto variables : %d", (count+a));
    return 0;
}
```

Storage Class	Lifetime	Scope	Initial Value	Storage
auto	Function Block	Local	Garbage	RAM Memory
static	All Program	Local	Zero	RAM Memory
extern	All Program	Global	Zero	RAM Memory
register	Function Block	Local	Garbage	CPU Registers

Constants in C

A constant is a value or variable that can't be changed in the program Execution.

In C programming language, a constant can be of any data type like integer, floating-point, character, string and double, etc.,

Integer constants

An integer constant can be a decimal integer or octal integer or hexadecimal integer.

Example

125 -----> Decimal Integer Constant
076 -----> Octal Integer Constant
0X3A -----> Hexa Decimal Integer Constant
50u -----> Unsigned Integer Constant
30l -----> Long Integer Constant
100ul -----> Unsigned Long Integer Constant

Floating Point constants

A floating-point constant must contain both integer and decimal parts. Some times it may also contain the exponent part. When a floating-point constant is represented in exponent form, the value must be suffixed with 'e' or 'E'.

Example The floating-point value **3.14** is represented as **3E-14** in exponent form.

Character Constants

A character constant is a symbol enclosed in a single quotation. A character constant has a maximum length of one character.

Example

'A'

'2'

'+'

String Constants

A string constant is a collection of characters, digits, special symbols and escape sequences that are enclosed in double quotations.

We define string constant in a single line as follows...

"This is programming"

We can define string constant using multiple lines as follows...

```
" This\  
is\  
programming "
```

We can also define string constant by separating it with white space as follows... **"This" "is" "programming"**

All the above three defines the same string constant.

Comments in C

Comments in C language are used to provide information about lines of code. It is widely used for documenting code. There are 2 types of comments in the C language.

1. Single Line Comments
2. Multi-Line Comments

Single Line Comments

Single line comments are represented by a double slash //. Let's see an example of a single line comment in C.

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

Output:

Hello C

Even you can place the comment after the statement. For example

1. `printf("Hello C"); //printing information`

Multi Line Comments

Multi-Line comments are represented by slash asterisk * ... *\ . It can occupy many lines of code, but it can't be nested. Syntax:

```
/* Multiple lines of code
   to be commented like this
*/
```

Let's see an example of a multi-Line comment in C.

```
#include<stdio.h>
int main() {
/*printing information
Multi-Line Comment*/
printf("Hello C");
return 0;
}
```

Output:

Hello C

Input/Output function in C

Input and output functions are divided into two types

Formatted Input/Output functions

Unformatted Input/output functions

Formatted I/O functions: These are used to take various inputs from the user and display multiple outputs to the user. These I/O supports all data types like int, float, char, and many more. These functions are called formatted I/O functions because we can use the format specified in these functions and hence, we can format these functions according to our needs.

Example Format Specifies: %d used for int and signed integer values

%f used for float and signed floating-point values

Formatted I/O functions are

1. printf()
2. scanf()
3. sprintf()
4. sscanf()

printf():

printf() function is used in a C program to display any value like float, integer, character, string, etc on the console screen. It is a pre-defined function defined in the **stdio.h** (header file).

Syntax 1: To display any variable value.

printf("Format Specifiers", var1, var2,, varn);

Example:

```
#include <stdio.h>
int main()
{
    int a;
    a = 20;
    printf("%d", a);
    return 0;
}
```

Output

20

Syntax 2: To display any string or a message

printf("Enter the text which you want to display");

Example:

```
#include <stdio.h>
int main()
{
    printf("This is a string");
    return 0;
}
```

Output

This is a string

scanf():

It is used in the C program for reading or taking any value from the keyboard by the user, these values can be of any data type like integer, float, character, string, and many more. This function is defined in **stdio.h** (header file), so it is also a pre-defined function. In scanf() function we use **&** (address-of operator) which is used to store the variable value on the memory location of that variable.

Syntax:

scanf("Format Specifier", &var1, &var2,, &varn);

Example:

```
#include <stdio.h>
int main()
{
    int num1;
    printf("Enter an integer number: ");
}
```

```
scanf("%d", &num1);  
printf("You have entered %d", num1);  
return 0;  
}
```

Output

Enter an integer number: 50
You have entered 50

sprintf():

sprintf stands for “**string print**”. This function is similar to printf() function but prints the string into a character array instead of printing the string on the console screen. It is a pre-defined function defined in the **stdio.h** (header file).

Syntax:

```
sprintf(array_name, “format specifier”, variable_name);
```

Example:

```
#include <stdio.h>  
int main()  
{  
    char str[50];  
    int a = 2, b = 8;  
    sprintf(str, "%d and %d are even number", a, b);  
    printf("%s", str);  
    return 0;  
}
```

Output

2 and 8 are even number

sscanf():

sscanf stands for “**string scanf**”. This function is similar to scanf() function but it reads data from the string or character array instead from the console screen. It is a pre-defined function defined in the **stdio.h** (header file).

Syntax:

```
sscanf(array_name, “format specifier”, &variable_name);
```

Example:

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

```
char str[50];
int a = 2, b = 8, c, d;
sprintf(str, "a = %d and b = %d", a, b);
sscanf(str, "a = %d and b = %d", &c, &d);
printf("c = %d and d = %d", c, d);
return 0;
}
```

Output

c = 2 and d = 8

Unformatted Input/Output functions

Unformatted I/O functions are used only for character data type or character array/string and cannot be used for any other datatype. These functions are used to read single input from the user at the console and it allows display of the value at the console. These functions are called unformatted I/O functions because we cannot use format specifiers in these functions and hence, cannot format these functions according to our needs.

Unformatted I/O functions

1. `getch()`
2. `getche()`
3. `getchar()`
4. `putchar()`
5. `gets()`
6. `puts()`
7. `putch()`

getch():

`getch()` function **reads a single character** from the keyboard by the user but **doesn't display** that character on the console screen and immediately returns without pressing the <enter> key. This function is declared in **conio.h** (header file). `getch()` is also used to hold the screen.

Syntax:

`getch();` or `variable-name = getch();`

Example:

```
#include <conio.h>
#include <stdio.h>
int main()
{
    printf("Enter any character: ");
```



```
    getch();
    return 0;
}
```

Output:

Enter any character:

getche():

getche() function **reads a single character** from the keyboard by the user and **displays** it on the console screen and immediately returns without pressing the enter key. This function is declared in **conio.h** (header file).

Syntax:

getche(); or **variable_name = getche();**

Example:

```
#include <conio.h>
#include <stdio.h>
int main()
{ printf("Enter any character: ");
  getch();
  return 0;
}
```

Output:

Enter any character: g

getchar():

The getchar() function is used to read only a first single character from the keyboard whether multiple characters are typed by the user and this function reads one character at one time until the <enter> key is pressed. This function is declared in **stdio.h**(header file)

Syntax:

Variable-name = getchar();

Example:

```
#include <stdio.h>
int main()
{
  char ch;
  printf("Enter the character: ");
  ch = getchar();
  printf("%c", ch);
}
```

```
return 0;  
}
```

Output:

Enter the character: a

a

putchar():

The putchar() function is used to display a single character at a time by passing that character directly or by passing a variable that has already stored a character. This function is defined in **stdio.h** (header file)

Syntax:

```
putchar(variable_name);
```

Example:

```
#include <stdio.h>  
int main()  
{  
    char ch;  
    printf("Enter any character: ");  
    ch = getchar();  
    putchar(ch);  
    return 0;  
}
```

Output:

Enter any character: Z

Z

gets():

gets() function reads a group of characters or strings from the keyboard by the user and these characters get stored in a character array. This function allows us to write space-separated texts or strings. This function is declared in **stdio.h** (header file).

Syntax:

```
char name[length of string in number]; //Declare a char type variable of any length  
gets(name);
```

Example:

```
#include <stdio.h>
int main()
{ char name[50];
  printf("Please enter some texts: ");
  gets(name);
  printf("You have entered: %s", name);
  return 0;
}
```

Output:

Please enter some texts: C Program
You have entered: C Program

puts():

In C programming puts() function is used to display a group of characters or strings which is already stored in a character array. This function is declared in **stdio.h** (header file).

Syntax:

```
puts(identifier_name );
```

Example:

```
#include <stdio.h>
int main()
{
  char name[50];
  printf("Enter your text: ");
  gets(name);
  printf("Your text is: ");
  puts(name);
  return 0;
}
```

Output:

Enter your text: C Program

Your text is: C Program

putch():

PTC UNIT-1

putch() function is used to display a single character which is given by the user and that character prints at the current cursor location. This function is declared in **conio.h** (header file)

Syntax:

```
putch(variable_name);
```

Example:

```
#include <conio.h>
#include <stdio.h>
int main()
{
    char ch;
    printf("Enter any character:\n ");
    ch = getch();

    printf("\nEntered character is: ");
    putch(ch);
    return 0;
}
```

Output:

```
Enter any character:
Entered character is: d
```

Formatted I/O vs Unformatted I/O

S No.	Formatted I/O functions	Unformatted I/O functions
1	These functions allow us to take input or display output in the user's desired format.	These functions do not allow to take input or display output in user-desired format.
2	These functions support format specifiers.	These functions do not support format specifiers.
3	These are used for storing data more user friendly	These functions are not more user-friendly.
4	Here, we can use all data types.	Here, we can use only character and string data types.
5	printf(), scanf, sprintf() and sscanf() are examples of these functions.	getch(), getche(), gets() and puts(), are some examples of these functions.

Escape Sequences:

PTC UNIT-1

To display the output in different lines or as we wish, we use some special characters called **escape sequences**. Escape sequences are special characters with special functionality used in printf() function to format the output according to the user's requirement. In the C programming language, we have the following escape sequences

Escape sequence	Meaning
\n	Moves the cursor to New Line
\t	Inserts Horizontal Tab (5 characters space)
\v	Inserts Vertical Tab (5 lines space)
\a	Beep sound
\b	Backspace (removes the previous character from its current position)
\\	Inserts Backward slash symbol
\?	Inserts Question mark symbol
\'	Inserts Single quotation mark symbol
\"	Inserts Double quotation mark symbol

Example:

```
#include<stdio.h>
#include<conio.h>
void main(){
    clrscr();
    printf("Welcome to\n");
    printf("Programming\n");
    printf("using C Languauge");
    getch()
}
```

Programming examples

1. Write a Program to Display "Hello, World!"

```
#include <stdio.h>

int main()
{
    // printf() displays the string inside quotation
    printf("Hello, World!");

    return 0;
}
```

Output

Hello, World!

2. Write a Program to Print an Integer

```
#include <stdio.h>

int main()
{
    int number;

    printf("Enter an integer: ");

    // reads and stores input
    scanf("%d", &number);

    // displays output
    printf("You entered: %d", number);

    return 0;
}
```

Output

Enter an integer: 25

You entered: 25

3. Write a Program to Add Two Integers

```
#include <stdio.h>

int main()
{
    int number1, number2, sum;

    printf("Enter two integers: ");

    scanf("%d %d", &number1, &number2);

    // calculating sum

    sum = number1 + number2;

    printf("%d + %d = %d", number1, number2, sum); return 0;
}
```

Output

```
Enter two integers: 12
11
12 + 11 = 23
```

4. write a Program to Find the Size of Variables

```
#include<stdio.h>

int main()
{
    int intValue;
    float floatValue;
    double doubleValue;
    long double ldoubleValue;
    char charValue;

    // sizeof evaluates the size of a variable

    printf("Size of int: %zu bytes\n", sizeof(intValue));
}
```

```
printf("Size of float: %zu bytes\n", sizeof(floatValue));
printf("Size of double: %zu bytes\n", sizeof(doubleValue));
printf("Size of long double: %zu bytes\n", sizeof(ldoubleValue));
printf("Size of char: %zu byte\n", sizeof(charValue));
return 0;
}
```

Output

Size of int: 4 bytes

Size of float: 4 bytes

Size of double: 8 bytes

Size of long double: 12 bytes

Size of char: 1 byte

5. Write a Swap Numbers Using Temporary Variable

```
##include<stdio.h>
int main()
{
    double first, second, temp;
    printf("Enter first number: ");
    scanf("%lf", &first);
    printf("Enter second number: ");
    scanf("%lf", &second);
    // value of first is assigned to temp
    temp = first;
    // value of second is assigned to first
    first = second;
    // value of temp (initial value of first) is assigned to second
    second = temp;
    // %.2lf displays number up to 2 decimal points
    printf("\nAfter swapping, first number = %.2lf\n", first);
    printf("After swapping, second number = %.2lf", second); return 0;
}
```

Output

Enter first number: 7.5

Enter second number: 20.9

After swapping, first number = 20.90

After swapping, second number = 7.50

6. Write a Program to Check Even or Odd

```
#include <stdio.h>
int main()
{
    int num;
    printf("Enter an integer: ");
    scanf("%d", &num);
    // true if num is perfectly divisible by 2
    if(num % 2 == 0)
        printf("%d is even.", num);
    else
        printf("%d is odd.", num);

    return 0;
}
```

Output

Enter an integer: -7

-7 is odd.

7. Write a Program to Print ASCII Value

```
#include <stdio.h>
int main()
{
    char c;
    printf("Enter a character: ");
    scanf("%c", &c);

    // %d displays the integer value of a character
    // %c displays the actual character
}
```

```
printf("ASCII value of %c = %d", c, c);  
return 0;  
}
```

Output

Enter a character: A

ASCII value of A = 65

8. Program to perform all 4 arithmetic operations

```
//Program to perform all arithmetic operations  
/*Input: Enter any two numbers: 20 5  
Output:  
SUM = 25  
DIFFERENCE = 15  
PRODUCT = 100  
QUOTIENT = 4.000000  
MODULUS = 0  
*/  
#include <stdio.h>  
int main()  
{  
    int num1, num2;  
    int sum, sub, mult, mod;  
    float div;  
  
    /* Input two numbers from user */  
    printf("Enter any two numbers: ");  
    scanf("%d%d", &num1, &num2);  
  
    /* Perform all arithmetic operations */  
    sum = num1 + num2;  
    sub = num1 - num2;  
    mult = num1 * num2;  
    div = (float)num1 / num2;  
    mod = num1 % num2;  
  
    /* Print result of all arithmetic operations */  
    printf("SUM = %d\n", sum);  
    printf("DIFFERENCE = %d\n", sub);  
    printf("PRODUCT = %d\n", mult);
```

```
printf("QUOTIENT = %f\n", div);
printf("MODULUS = %d", mod);
return 0;
}
```

9. Program to find perimeter of a rectangle using its Length & Breadth.

```
//Program to find perimeter of rectangle
```

```
/*
```

```
Input:
```

```
Enter length of the rectangle: 5
```

```
Enter width of the rectangle: 8
```

```
Output:
```

```
Perimeter of rectangle = 26.000000 units
```

```
*/
```

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

```
int main()
```

```
{
```

```
float length, width, perimeter;
```

```
// Input length and width of rectangle from user
```

```
printf("Enter length of the rectangle: ");
```

```
scanf("%f", &length);
```

```
printf("Enter width of the rectangle: ");
```

```
scanf("%f", &width);
```

```
/* Calculate perimeter of rectangle */
```

```
perimeter = 2 * (length + width);
```

```
/* Print perimeter of rectangle */
```

```
printf("Perimeter of rectangle = %f units ", perimeter);
```

```
return 0;
```

```
}
```

10. Program to find diameter, circumference and area of a circle using its radius.

```
/*Program to find Diameter, circumference and area of a circle
```

```
d=2*r
```

```
c=2*p*r
```

```
a=2*p*r*r
```

```
Input: Enter radius of circle: 5
```

```
Output:
```

```
Diameter of circle = 10.00 units
```

```
Circumference of circle = 31.40 units
```

```
Area of circle = 78.50 sq. units
```

```
*/
```

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

```
int main()
```

```
{
```

```
float radius, diameter, circumference, area;
```

```
/*
 * Input radius of circle from user
 */
printf("Enter radius of circle: ");
scanf("%f", &radius);

/*
 * Calculate diameter, circumference and area
 */
diameter = 2 * radius;
circumference = 2 * 3.14 * radius;
area = 3.14 * (radius * radius);

/*
 * Print all results
 */
printf("Diameter of circle = %.2f units \n", diameter);
printf("Circumference of circle = %.2f units \n", circumference);
printf("Area of circle = %.2f sq. units ", area);

return 0;
}
```

Output:

Enter radius of circle: 5
Diameter of circle = 10.00 units
Circumference of circle = 31.40 units
Area of circle = 78.50 sq. units