

# Full C Programming Course

By Dheeraj Patidar

## **\*\*Index\*\***

### **1)Introduction to C**

=> What is C Programming?

- Some Facts About C
- Applications of C
- Character Set of C

=> Why to learn or use C Programming

### **2) Basic Concepts in C**

=> Variable, Constant, Keywords in C

=> Structure of C Program ( header files, main function, variable Declaration, scanf, printf )

=> Flow Chart & Algorithm

=> Flow of program in computer

### **3) Datatype in C**

=> What is Data Type?

=> Classification of Data Type

=>"limits.h" header file

=>Size of operator

=> What is ASCII?

=> Why Char Datatype limits represent in integer?

=> Why any char Required 1 byte memory in computer?

### **4) Types of Instructions in C**

=> What are Instructions in C?

=>Types of Instructions in C

- Type declaration instruction (int,float,char)

- Arithmetic instruction or Operations (+, -, \*, /, %)
- Control instructions
  - 1) Sequence Control Instruction
  - 2) Decision Control Instruction (if, if-else, (Conditional operators(?, :)))
  - 3) Loop Control Instruction(while, do-while,for)
  - 4) Case-Control Instruction(switch, break, continue, goto)

## 5) Operators in C

=>What are Operators in C Programming?

=> Types of Operators in C

- 1) Arithmetic Operators
  - Modify Operators (increment/decrement)
- 2) Relational Operators
- 3) Logical Operators
- 4) Bitwise Operators
- 5) Assignment Operators
- 6) Misc Operators

=>Hierarchy of Operators / Operators Precedence Table

## 6) Functions & Recursion

=> What is Function (Syntax)

=> Parts of function (return type, identity, parameter, body)

=> Function Declaration

=> Calling a Function (call by value, call by reference)

=> Types of Functions in C ( Predefined Functions, User-defined functions)

=>Recursion

## 7) Pointers (\*,&)

=> What is Pointer? (Syntax, Declaration)

=>Types of Pointers (Typed, Untyped)

=>Pointer to Functions

=>Dangling Pointer

## 8)Array

=> What is Array? (Syntax, Declaration)

=> How Memory Will be Allocated in Array Consept?

=> How to Access Array Elements?

=> How to initialize Array?

- Methods=>Direct initialisation using Curly Braces, By assigning Values one by one, and using Loops)

=> How to process the elements of Array?

=> Local & Global Declaration of Array

=> Types of Array => (1d array, 2d array, md array)

=> Pointer to Array

## **9) String**

=> What is String? (Syntax, Declaration)

=> How Memory Will be Allocated in String?

=> How to initialise a string?

=> How to Access the Elements of String?

=> How to Process the Elements of String?

=> string Handling Functions from "string.h" Header file:-  
gets(), puts(), strlen(), strrev(), strcpy(), strcat(), strcmp().

## **10) Structure**

=>What is Structure?

=> How Memory Will be Allocated in Structure?

=> How to Access the Members of Structure? (Accessor, Arrow Operator)

=> How to initialize Structure?

=> Size of Structure by sizeof() Operator (by variable, by struct type)

=> Local and global structure

=> Pointer to Structure

=> Array to structure

## **11) Union**

=> What is Union in C? (Syntax, Declaration)

=> How memory will be allocated in union concept?

=> Size of union

=> How to access the members of union? (Accessor, Arrow Operator)

## **12) Dynamic Memory Allocation**

=> What is static memory allocation?

=> Some problems in static memory allocation?

=> What is dynamic memory allocation?

=> How to allocate memory dynamically? (malloc, calloc, realloc, free)

=> Different b/w Dynamic & Static Memory?

## **13) File I/O**

=> What is data?

=> What is data persistence?

=> What is file and uses of file?

=> What is file handling **or** file I/O

=> What is file pointer?

=> Basic file operations in C programming:-

1. Opening / Creating a file >> (fopen) (Syntax, Declaration)
  - Opening modes in File I/O
2. Closing a file >> (fclose) (Syntax, Declaration)
3. Reading a file (fgetc, fscanf)
4. Writing a file (fprintf, fputc)

## **14) typedef**

=> What is typedef? (Syntax, Declaration)

=> Examples of Typedef in:-

- Primitive,
- Derived,
- User Defined dataType

## **15) Storage Classes**

=> What represent storage class about variable?

=> Types of Scopes? (Block Scope, Method Scope, Program Scope)

=> Types of storage classes in C:-

- Automatic storage class (auto)
- Register storage class (register)

- Static storage class (static)
- External storage class (extern)

## 16) Command Line Arguments

=>What is Command Line Arguments?

=> Who call main() function?

=> What are the different ways of run any C program?

(Using IDE, By Double Click, Command Line)

=> Components of command line Arguments (int **argc**, char **\*argv[]**)

=> Use of atoi(), and atof().

## 17) Graphics in C

=>What is Graphics in C?

=>How to initialise graphic mode in C? (initgraph())

=> How to close graphic mode in C? (closegraph())

=> Some more essential functions in graphic mode:-

1. Shapes related functions in Graphic mode  
(circle, bar, rectangle, line, ellipse)

2. Colors related functions in Graphic mode  
(setcolor, setbkcolor)

=> Text Related Functions in graphic Mode  
(outtextxy, settextstyle)

=>function to Clear Screen in Graphic Mode  
cleardevice();

\*\*\*\*\*

# Introduction to Programming Language

## Q. What is Language and Computer Programming Language?

**Ans.=>** Language is a set of instruction or rules (grammar) by the help of which we can communicate to each other. For example we use Hindi and English language for communication.

We all know that computer is an electronic dump machine. To communicate with computer we need to use **programming languages or Computer Languages** like c,c++,java, python,etc...

### Need of Computer Languages:-

We need computer languages to develop or create programs, after combining n number of programs one software is developed. Each of the program of software performing a particular task and these programs are developed through different different programming languages or may be same.

### Why we Develop Programs & Softwares?

We create programs and softwares to solve particular problem in quick manner. The working speed of computer is much faster than human being that's why we develop softwares and programs for computer to perform any task in quick manner.

## Q. What types of applications (Software) we can develop using programming languages?

**Ans=>** we can develop 2 types of application using programming languages :-

**i) Standalone Applications:** the applications which must be installed in computer. This type of applications only **compatible** for single operating system. The application which are compatible for one operating system is not compatible for another operating system. These applications are always dependent to operating system.

For Example:

**windows operating system** having there own MS word, MS office, video player, audio player, etc

In another hand **MAC operating system** having their on MS word, MS office, video player, audio player, etc. We cannot use applications of one operating

system into another operating system. That's why this type of application is called **standalone application**.

**ii) Web Application:** the applications which can be used without installing in computer is called web applications. It is not mandatory to download this type of applications in the computer. These applications are independent (Not depend) to any operating system.

We can use these type of Applications in Any browser. There is no need to install them. **standalone version** of these **web applications** are also available but it is our **choice** to download them and to use them by browser.

For Example:

**WhatsApp, Facebook, YouTube, Gmail**, and all other **social applications** can be used in computer without installing and these applications are comes under web applications. In this type of application some **payment transactions web applications** are also include like icici bank.

### **Why we are not using Standalone Applications in Another Operating System?**

Because different different operating system having there own Extensions just like files. It is not possible to open .txt extension file in any mp3 player.because mp3 player can only understand .mp3 extension. The file having .txt extension can only be open in any notepad.

Just in same way we cannot use windows operating system applications in another operating system like MAC OS. Because windows application having .exe extension.

Windows OS ==> .exe,

MAC OS ==> .dmg,

Linux OS ==> .rpm, .tar, etc.

**Note:** *C and C++ is used to Develop only Standalone Applications thats why major parts of different different OS is Developed through C and C++. In other hand through all other Programming Languages we can develop either web application or Standalone Applications.*

\*\*\*\*\*

# 1.

## Introduction to C

### **Q. What is C Programming?**

**Ans=>** C is a programming language developed at Bell telephone laboratories in 1972 by Dennis Ritchie. It is one of the oldest and finest programming language. C programming language is reliable simple and easy to use structured programming language. Major parts of popular operating system like windows, Unix, Linux, etc are written in C. it is the most appropriate language for learning computer programming.



C programming language is high level language and in another high level language the Syntax rule of C language is used hence it is the basic language for all other high level languages.

### **Some Important Facts about C :-**

- C is invented to Write Linux operating system.
- C is a successor of B language which was introduced around 1970.
- C was formalized in 1988 by the American National standard Institute (ANSI)
- Today's most popular Linux OS and RDBMS MySQL have been written in C
- She was originally first implemented on the DEC PDP-11 computer in 1972.

### **Applications of C:-**

C was initially used for system development work, particularly the programs that make-up the operating system. It was initially used in System Development because it produces code that runs nearly the code of assembly language. Some examples of the use of C are -

Operating Systems, Language Compilers, Assemblers, Text Editors, Print Spoolers, Network Drivers, Modern Programs, Databases, Language Interpreters Utilities

### **Character Set of C Programming :-**

1. Alphabets => A to Z, a to z.
2. Digits => 0 to 9.
3. Special Symbols => ~ ! @ # % ^ & \* ( ) \_ - + : \ { } [ ] ; , < > . ? / \$ " ' "

### **Q. Why to Learn or use C Programming**

**Ans=>** we learn or use C Programming Because of Some Features and Advantages of C :-

- Easy to learn
- Structured language
- It produces efficient programs

- It can handle low-level activities
- It can be compiled on a variety of computer platforms

\*\*\*\*\*

## 2.

# Basic Concepts of C

### Variable, Constant, & Keywords in C

**=>Variables :-** Variables are the storage place where we can store a number of Values. We can store Data in computer through Variables. To store any value in computer we need to help variables. Through variables we can easily Store values in computer and also accessing them easily.

Ex.=> a=2, b='a', c=2.5

### Rules for Declaring or Naming Variables in C programming :-

1. The first character must be an alphabet or underscore(\_).
2. No commas, blanks allow.

3. No special symbol other than underscore is allowed.
4. Variable names are case sensitive

**=>Constants :-** An entity whose value doesn't change is called a constant. if we change value of constant during execution of program then we get compilation error. Constants are the values which are stored in Computer thought Variables. Means variables holds constant values in them.

### Types of constant

**Primarily there are 3 types of constant :-**

- Integer constant = -1,6,7,9
- Real constant = -322.1,2.5,7.0
- Character constant = 'a','\$','@'(must be enclosed within single inverted commas).

**=>Keywords :-** keywords are predefined reserved words in a programming language. these keywords help us to use the functionality of C programming. Due to their special meaning we cannot be used them for other purpose.

**There are 32 keywords available in C programming :-**

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

### **\*\*\*The basic structure of a C program\*\*\***

All c programs have to follow a basic structure. A c program starts with the main function and executes instructions presents inside it. Each instruction terminated with a semicolon(;).

**There are some basic rules which are applicable to all the c programs :-**

1. Every program's execution starts from the main function.
2. All the statements are terminated with a semi-colon.
3. Instructions are case sensitive.
4. Instructions are executed in the same order in which they are written.
5. For printing output of any C Program we Need to use predefined function of **stdio.h** header file Called printf( ).
6. For taking input there is one more predefined function of **stdio.h** header file is used called scanf().

#### **Program to add Two numbers :-**

/* Program to add two Numbers */	----> <b>Comment</b>
#include<stdio.h>	----> <b>Header File with Preprocessor</b>
void main()	----> <b>Main Function</b>
{	----> <b>Function Body Starting</b>
int a,b,c;	----> <b>Variable Declaration</b>
printf("\n Enter two Numbers");	----> <b>Output Statement for Taking Input</b>
scanf("%d%d",&a,&b);	----> <b>Input Statement</b>
c=a+b;	----> <b>Arithmetic Operation / Processing</b>
printf("\nSum= %d",c);	----> <b>Output Statement for Output Printing</b>
}	----> <b>Function Body End</b>

#### **Explanation of Above Program Structure :**

**Comments :-** Comments are used to understand program in better way.

Comments are not executed by Compiler. Comment are not part of program.

They are used for just small note about program Writer in program to understand program clearly. We can add comment anywhere in c program after terminating Instruction using semicolon.

**There are 2 types of comment available in C:-**

- **Single line comment:** //This is a comment.
- **Multi line comment:** /\*This is multi line comment\*/

**Header files:** In C programming collection of header files is called library and ***collection of related predefined functions is called Header files.*** To use any predefined function in c program we need to to declare related header file of this function before main function.

C library is consist of n number of header files and these header files having n number of related predefined functions. Each of the function in different different header file having special functionality to perform particular task.

There are n numbers of Preprocessors available in C and one of them is **#include Preprocessor**. With the help of **Preprocessor #include** we can Declare any header file in c program.

Ex.=> **stdio.h header file** contains n number of functions like printf(), scanf(), fprintf(), fscanf(), fopen(), fgets(), fputs(), etc...

**Some Header files and their Functions :-**

**math.h =>** pow(), rand(), sqrt(), etc..

**graphics.h =>** setcolor(), setbkcolor(); circle(), bar(), line(), outtextxy(), textstyle(), etc...

**string.h =>** strlen(), strrev(), strcat(), strcpy(), strcmp(), etc...

**conio.h =>** clrscr(), getch(), getche(), textcolor(), textbackground(), etc...

**main function:** main function is a predefined function in c programming. This function is compulsory to to start writing any program because C program execution is starting from main function. Without main function execution of program is not possible.

We can also create other functions like main function but all other functions are calling **directly or indirectly** from main function. But main function is not calling from inside to program because it is a predefined function which is calling by operating system. We can pass arguments to main function by command line.

**Variable Declaration:** we all know that to store any value in computer we need to help variables. Through variables we can easily Store values in computer and also accessing them easily. In the declaration of a variable it is mandatory to specify its data type. In C programming some predefined primitive data types are available to declare variables. These are int, float, char, and void.

- int variable can hold one integer value in it
- float variable can hold one real (floating) value in it.
- char variable can hold one character value in it.
- void variable can hold one value of any type.

**printf function:** printf function is a predefined function of **stdio.h** header file. It is used in C program in 2 ways:-

1. To take value from End-user.
2. To print output of program.

Prototype or Syntax of printf function in 2 ways:-

1. For Requesting input from End-user:

```
printf("request statement (char*string)");
```

2. For Printing output of program:

```
printf("ouput statement (char*string)", variables list);
```

**scanf function:** scanf is also predefined function of **stdio.h** header file which is used to store value given by End-user through printf function. To store value in variables we need address of operator (&). We can store n number of values in n Number of variables through scanf using **format specifies** and using **address of operator (&)** before variable name.

Prototype or Syntax of scanf function:-

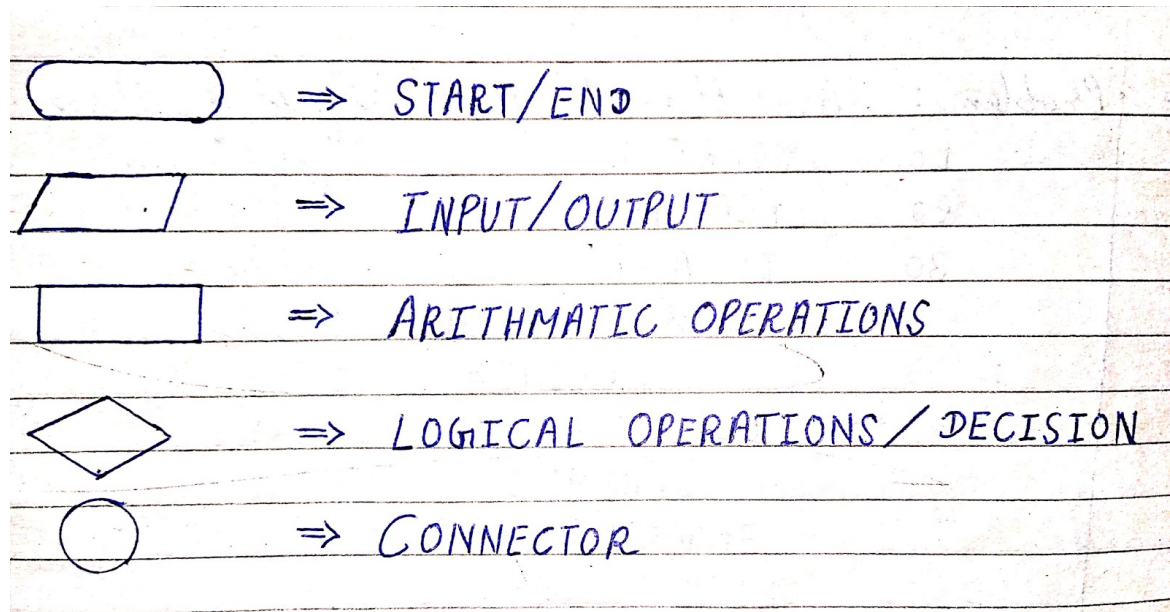
```
scanf ("formate specifiers",&variable1, &variable2,...);
```

```
Ex=> scanf("%d%f%c",a,b,c);
```

### \*\*\*Flow Chart & Algorithm\*\*\*

**Flow Chart** => Flow Chart is a Pictorial or Diagrammatic Representation of Institutions of programs. With the help of flowchart we can easily understand flow of any program. flowchart uses various symbols to represent different operations involved in the solution of a program.

Following symbols are used to draw flow Chart of any program :-



**Algorithm** => algorithm is a set of instruction to perform a given task in a definite time, written in normal English language is known as algorithm. Algorithm is a step-by-step procedure which define flow of execution of instructions one by one to produce desired output.

#### **Characteristics of an Algorithm :-**

**Unambiguous:-** All the Steps Should be Clear and lead to be one meaning.

**Input:-** should have 0 or more well defined inputs

**Output:-** should have 1 or more well defined outputs.

**finiteness:-** must be terminated after a finite number of steps.

**Feasibility:-** Algorithm Should be Correct with Available Resources.

**Independent:-** it should be independent of any programming language.

### **\*\*\*Flow of Program in C\*\*\***

**Step 1:** C program (source code) is sent to preprocessor first. The preprocessor generates an expanded source code.

**Step 2:** Expanded source code is sent to compiler which compiles the code and converts it into assembly code.

**Step 3:** The assembly code is sent to assembler which assembles the code and converts it into object code. (.o or .obj).

**Step 4:** The object code is sent to linker which links it to the library such as header files. Then it is converted into executable code. (.exe)

**Step 5:** The executable code is sent to loader which loads it into memory and then it is executed. After execution, output is sent to console.

**\*\*\*\*\***



# 3.

## Data Types in C

### What is Data Types?

data type is nothing but it is a representation of a data. In the declaration of any variable it is mandatory to specify its data type.

**Syntax of Data type:-** data\_type variable\_name;

Ex.=> int a;

### It represents two things:-

- how much memory is required to allocate for variable.
- second is what type of data is allowed to store in a particular variable.

For example=> In "int a" variable integer data type is allowed and it required 2 or 4 bytes for memory allocation.

### Classifications of Data Types in C

- **Primitive Data Type**

1. int

>short int : 2 bytes memory required

>>signed short : -32768 to 32767

>> unsigned short : 0 to 65535

> int : 2 or 4 bytes memory required

>> signed int : -2,147,483,648 to 2,147,483,647

>> unsigned int : 0 to 4,294,967,295

> long int : 4 bytes memory required  
>> signed long int : -9223372036854775808 to  
9223372036854775807  
>> unsigned long int : 0 to 18446744073709551615

2. float

> float : 4 bytes memory required [1.2E-38 to 3.4E+38]  
> double : 8 bytes memory required [2.3E-308 to 1.7E+308]  
> long double : 10 bytes memory required [2.3E-308 to 1.7E+308]

3. char : 1 bytes memory required

>signed : -128 to 127  
>unsigned : 0 to 255

4. void

● **Derived Data Type**

1. Array
2. string
3. function
4. pointer

● **User Defined Data Type**

1. struct
2. union
3. typedef
4. enum

**Some Important Questions about Char Data Type :-**

**Q. What is ASCII ?**

**Ans=>** ASCII stands for American Standard Code for International Interchange. This ASCII System Consist of Total 256 (0 to 255) Characters. Through ASCII System we can Easily Access Characters in Computer.

ASCII System Having Number from 0 to 255 for Each if the Character. This system is Applied for All Programming Languages.

**Q. Why Char Data Type Range or Limit represent in integer?**

**Ans=>** Because we are representing a character using Character system Called **ASCII**. ASCII Consist of total 256 (0 to 255) character. Due to which each character having perticular number. We Can Say that it is a Predefined Enumeration of All Characters in ASCII for All Computer Programming Languages.

**Q. Why Any Character Required 1 byte memory in Computer?**

**Ans=>** character Required 1 Byte memory for Allocation Because total characters in any programming language is definitely not more than 256. Means 256 is equal to  $2^8$  and  $2^8$  is 1 byte. That's why one character Required 1 Byte memory in Computer.

**\*\*\*limits.h Header File\*\*\***

**Q. How to determine limits of data types in C programming?**

**Ans=>** we can determine limits of data type through predefined global variables of "**limits.h**" header file. limits.h header file contains n number of predefined global variables. They are also called **macros of limit.h header file**. All these n number of global variables of limits.h having Constant values means we cannot modify them.

All these predefined global variables of limits.h header file are as follows:-

Macros /Variables	Values	Description
CHAR_BIT	8	Define the number of bits in a byte.
SCHAR_MIN CHAR_MIN	-128	Minimum value of a signed char.
SCHAR_MAX	127	Maximum value of a

CHAR_MAX		signed char.
UCHAR_MAX	255	Maximum value of an unsigned char.
SHRT_MIN	-32768	Minimum value of a short int
SHRT_MAX	32767	Maximum value of a short int
USHRT_MAX	65535	Maximum value of an unsigned short int
INT_MIN	-2147483648	Minimum value of an int
INT_MAX	2147483647	Maximum value of an int
UINT_MAX	4294967295	Maximum value of an unsigned int
LONG_MIN	-9223372036854775808	Minimum value of long int
LONG_MAX	9223372036854775807	Maximum value of long int
ULONG_MAX	18446744073709551615	Maximum value of an unsigned long int

### \*\*\*sizeof() operator\*\*\*

**Q. What is sizeof operator and what is the use of it in C programming?**

**Ans=>** size of operator is a unary operator which is used to determine the size of any operand. It returns this size of a variable. It can be applied to any data type like int, float, char, pointer, structure, typedef, union, etc.

Syntax or Prototype :- int sizeof(operand or data type)

Ex. => sizeof(a) , sizeof(int), sizeof(struct emp), sizeof(arr), sizeof(str).

\*\*\*\*\*

## 4. Instructions in C

### Q. What is Instructions in C?

**Ans.**=> Instructions are the commands given to program for compiler to do certain specific tasks. Without instructions there is no programming language. Each of the statement of any program is part of instructions.

There are three types of instructions available in C programming:-

- 1. Type Declaration Instruction :** this instruction is used to declare data type of variables. We all know that before writing any variable in program it is mandatory to specify its data type.
- 2. Arithmetic Instructions:** arithmetic instructions are those statements of c program by the help of which program perform mathematical operations like multiplication, addition, subtraction, division, etc. To perform mathematical operations in C program we need some operators (+, -, \*, /, %).
- 3. Control Instructions:** control instructions are those predefined statements in C program which are used to perform Special tasks like decision making. On the basis of different different Special tasks control instructions are further classified into four types :-

1) **Sequence control instruction:** sequence control instruction ensure that the C program is executed in the same order in which they written.

2) **Decision Control Instruction:** Decision control Instructions allow the computer to take a decision as to which instruction is to be executed next. Decision control instruction works on the basis of Given Condition (true/false).

Decision control instructions includes:-

- **if-else statement:** It is used to execute next Instruction of program on the basis of Condition. If the condition is true, then the if block will be executed, otherwise, the else block will be executed. In if-else statement else block is not necessary but optional. ***On the basis of requirements we can also use nested if, nested if else and else if logic in the C program.***

**Syntax:**

```
if ( condition to be checked) {  
    Statements-if-condition-true ;  
}  
else{  
statements-if-condition-false ;  
}
```

**Example of if-else:**

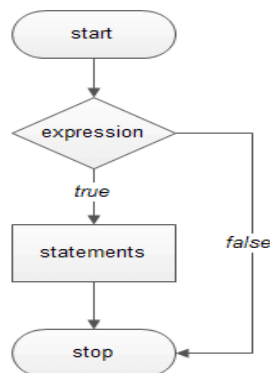
```
int a=10; //we can input value from end user also
if(a=11){
    printf("I am 11");
}
else{
    printf("I am not 11");
}
```

### Shorthand syntax for if-else:- (Conditional Operators)

Condition ? expression-if-true : expression-if-false;

**Ex=>** `a>1 ? printf("Greater") : printf ("Smaller");`

### Flow Chart of if-else:



- 3) Loop Control Instructions:** Loop control statements in C are used to perform looping operations until the given condition is true. Control comes out of the loop statements once condition becomes false. There are 3 types of loop control statements in C:-

**i) while loop:** In this Loop block keeps executing as long as the condition is true. It will be terminate after condition become false. It check the condition before executing the loop body. We can also use **Nested while loop** according to the requirement. *"If the condition never becomes false, the while loop keeps getting executed. Such a loop is known as an **infinite loop**".*

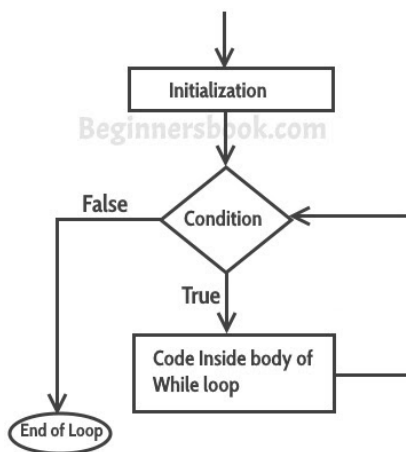
**Syntax:**

```
while(condition) {  
    statement(s);  
}
```

**Example:**

```
int i=0; //initialisation  
while (i<10){  
    printf("The value of i is %d",i); i++;  
}
```

**Flow Chart of while Loop:**



**ii) do while loop:** Do-while loop works very similar to while loop. But the main difference between while and do-while

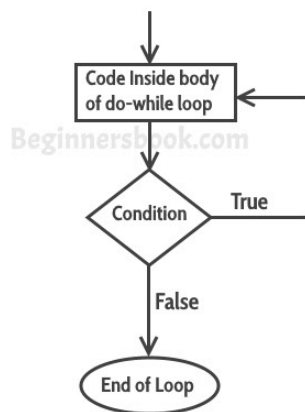


loop is that "while loop first check the condition & then execute block" and "do-while first executes the block & then checks the condition. do while loop execute at least once.

### **syntax of do-while:**

```
do {  
    statement(s);  
}while(condition)
```

### **Flow chart of do-while loop:**



**iii) for loop:** This Loop is also works same as while loop but only syntax is Different. while and do-while only takes condition for working but for loop taking 2 arguments along with Condition, first is initialisation, second is condition, and third one is increment or decrement operator.

### **syntax:**

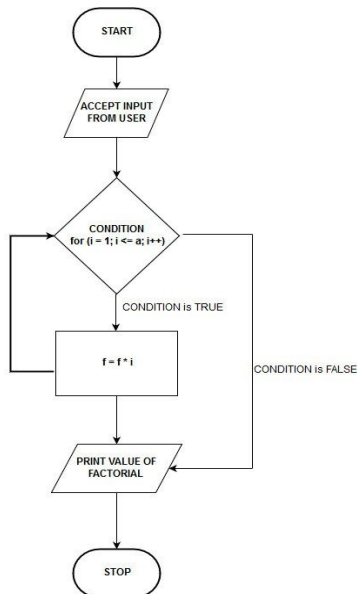
```
for ( init; condition; increment or decrement operators) {  
    statement(s);  
}
```

```
}
```

### Example:

```
for(i=0; i<3; i++)  
{  
    printf("%d",i);  
    printf("\n");  
}
```

### Flow chart of for loop:



**4) Case Control Instructions:** The statements which are used to execute only specific block of statements in a series of blocks are called case control Instructions.

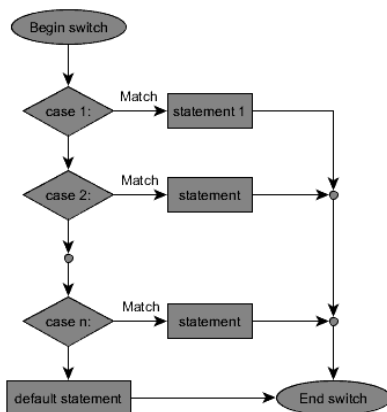
There are 4 Case Control Instructions available in c:-

**i) switch:** Switch case statements are used to execute only specific case statements based on the switch expression.

**Syntax=>**

```
switch (expression)
{
    case label1: statements;
                break;
    case label2: statements;
                break;
    case label3: statements;
                break;
    default:    statements;
                break;
}
```

**Flow chart of switch:**



**ii) break:** Break statement is used to terminate the while loops, switch case loops and for loops from the subsequent execution. Whatever the condition loop is terminated after using break statement.

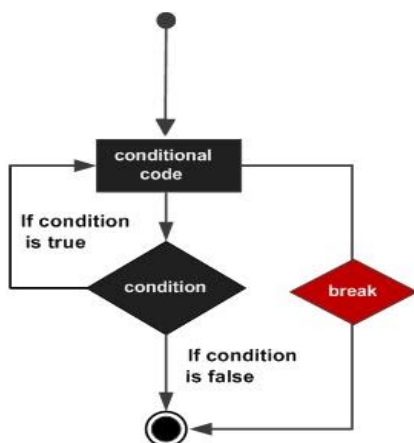
**Syntax:** break;

**Example:**

```
for (i=0; i<1000; i++){  
    printf("%d,",i);  
    if (i==5){  
        break;  
    }  
}
```

Output: 1,2,3,4,5

**Flow chart of break:**



**iii) continue:** Continue statement is used to continue the next iteration of for loop, while loop and do-while loops. Whatever the statements below **continue** will skip and control comes to next iteration of loop.

**Syntax:** continue;

**Example:**

```
for(i=0;i<10;i++)
```

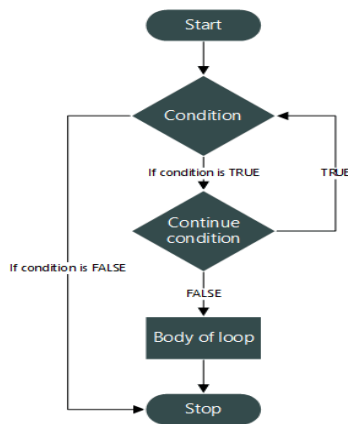
```

{
    if(i==5 || i==6)
    {
        continue;
    }
    printf("%d ",i);
}

```

Output: 0,1,2,3,4,7,8,9

### Flow Chart:

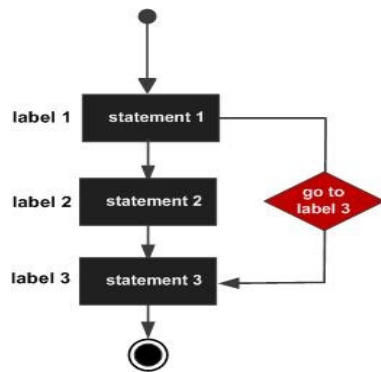


**iv) goto:** this case control statement is used to move control of program to particular statement. goto statement is used to transfer the normal flow of a program to the specified label in the program means control move to particular statement where we want. we can say that goto statement is used for customise flow of Program according to us. it is used for unconditional jump to labled statement.

### Syntax:

```
goto label;  
..  
.  
label: statement;
```

**flow chart of goto:**



\*\*\*\*\*

## 5. Operators in C

**Q. What are Operators in C Programming?**

**Ans=>** operators are the special symbols that tells the compiler to perform specific mathematical or logical operations.

**>>>Types of Operators in C**

On the basis of different different functionality, operators are classified into 6 type in C programming:-

### 1) Arithmetic Operators :- ( int a= 10, b=20;)

Operator	Description	Example
+	Add two operands	$a + b = 30$
-	Subtracts second operand from the first.	$a - b = -10$
*	Multiplies both operands.	$a * b = 300$
/	Divides numerator by denominator	$b / a = 2$
%	divide numerator by denominator and return remainder.	$b \% a = 0$
++	Increment operator increases the integer value by one. (called modify operator)	Pre increment $++a = 11$  Post increment $a++ = 11$
--	Increment operator decreases the integer value by one. (Called modify operator)	Pre decrement $--a = 9$  Post Decrement $a-- = 9$

### Modify Operators in C

Modify Operators are also called unary operator because they perform operation on single operands at a time.

There are 2 types of modify Operators available in C :-

- **Increment Operator:** used to increase integer value by one.

There are 2 types of increment:-

- i) **pre-increment:** printing value after increment.
- ii) **post-increment:** printing value before increment.

- **Decrement Operator:** used to decrease integer value by one.

There are 2 types of decrement:-

- i) **pre-decrement:** printing value after decrement.
- ii) **post-decrement:** printing value before decrement.

**Process of execution of modify Operators in C program step by step:-**

**Step1:** Pre-increment / Pre-decrement

**Step2:** Substitution

**Step3:** Evaluation

**Step4:** Assignment

**Step5:** Post-increment / Post-decrement

**2) Relational Operators:** (int a=10, b=20;)

Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(a == b) is not true



!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(a != b) is true
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	(a > b) is not true
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	(a<b) is true
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	(a>=b) is true
<=	Checks if the value of left operand is less than or equal to	(a<=b) is nit true

	the value of right operand. If yes, then the condition becomes true.	
--	--	--

### 3) Logical Operators: (int a=1, b=0)

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	!(A && B) is true.

	operator will make it false.	
--	------------------------------	--

**4) Bitwise Operators:** In arithmetic-logic unit (which is within the CPU), mathematical operations like: addition, subtraction, multiplication and division are done in bit-level. To perform bit-level operations in C programming, bitwise operators are used.

There are 6 bitwise Operators available in c programming :-

let (int a=12, b=25;)

Operators	Description	Example
<b>&amp;</b>	The output of bitwise AND is 1 if the corresponding bits of two operands is 1.  {1&1=1, 1&0=0, 0&0=0, 0&1=0.}	12 = 00001100 25 = 00011001  00001100 & 00011001  <u>00001000</u> = 8
<b> </b>	The output of bitwise OR is 1 if at least one corresponding bit of two operands is 1.	12 = 00001100 25 = 00011001  00001100   00011001  <u>          </u>

	{1&1=1, 1&0=1, 0&0=0, 0 &1= 1}	00011101 = 29
<b>^</b>	<p>The result of bitwise XOR operator is 1 if the corresponding bits of two operands are opposite.</p> <p>{1&amp;1=0, 1&amp;0=1, 0&amp;0=0, 0&amp;1=1}</p>	<p>12 = 00001100 25 = 00011001</p> <p>00001100 ^ 00011001</p> <hr/> <p>00010101 = 21</p>
<b>~</b>	<p>Bitwise compliment operator is an unary operator (works on only one operand). It changes 1 to 0 and 0 to 1.</p>	<p>35 = 00100011</p> <p>~ 00100011</p> <hr/> <p>11011100 = 220</p> <p>2's complement of 220</p> <p>-(11011100 + 1) -(11011101) -(36)</p> <p>Output on console is -36.</p>
<b>&lt;&lt;</b>	<p>Left shift Bitwise operator shifts all bits towards left by a certain number of specified bits.</p>	<p>212&lt;&lt;7</p> <p><b>Steps:-</b></p> <p>1) Decimal to binary 212 =</p>

		<p>11010100</p> <p>2) Sifting each bit by 7 in left= 00000001</p> <p>3) Binary to decimal 00000001= 1</p> <p>Output is 1</p>
>>	<p>Right Shift Bit wise operator shift all butts towards right by a certain number of specific bits</p>	<p>a&gt;&gt;2 Or 12&gt;&gt;2</p> <p><b>Steps:-</b></p> <p>4) Decimal to binary 12 = 1100</p> <p>5) Sifting each bit by 2 in rightt= 0011</p> <p>6) Binary to decimal 0011 = 3</p> <p>Output is 3</p>

## 5) Assignment Operators:

Operator	Description	Example
<b>=</b>	<b>"Simple assignment"</b> operator. Assigns values from right side operands to left side operand	$C = A + B$ will assign the value of $A + B$ to $C$
<b>+=</b>	<b>"Add AND assignment"</b> operator. It adds the right operand to the left operand and assign the result to the left operand.	$C += A$ is equivalent to $C = C + A$
<b>-=</b>	<b>"Subtract and assignment"</b> operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	$C -= A$ is equivalent to $C = C - A$
<b>*=</b>	<b>Multiply and assignment</b> operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	$C *= A$ is equivalent to $C = C * A$

<b>/=</b>	<b>Divide and assignment</b> operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A
<b>%=</b>	<b>Modulus and assignment</b> operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C % A
<b>&lt;&lt;=</b>	<b>Left shift and assignment</b> operator.	C <<= 2 is same as C = C << 2
<b>&gt;&gt;=</b>	<b>Right shift and assignment</b> operator.	C >>= 2 is same as C = C >> 2
<b>&amp;=</b>	<b>Bitwise AND and assignment</b> operator.	C &= 2 is same as C = C & 2
<b>^=</b>	<b>Bitwise exclusive OR and assignment</b> operator.	C ^= 2 is same as C = C ^ 2
<b> =</b>	<b>Bitwise inclusive OR and</b>	C  = 2 is same as C = C   2

	<b>assignment</b> operator.	
--	--------------------------------	--

## 5) Misc Operators: (sizeof & ternary)

Operator	Description	Example
<b>sizeof()</b>	Returns the size of a variable or data types.	int a; sizeof(a), where a is integer, will return 4.
<b>&amp;</b>	Returns the address of a variable.	&a; returns the actual address of the variable.
<b>*</b>	Pointer to a variable.	*a;
<b>? :</b>	Conditional Expression.	Used in Short Hand if statement:  Condition ? expression-if-true : expression-if-false;

### \*\*\*Operator Precedence in C\*\*\*

Operators priority or Operators Precedence determine in C program that which operator execute first in any statement. This term is also called hierarchy of operators. According to the statement condition Hierarchy of operators decides that which operator having higher priority for execution and which operator having lower priority in execution.



Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom.

Priority b/w Some Basic Operator:-

<b>Category</b>	<b>Operators</b>	<b>Associativity</b>
Not operator	<b>!</b>	Right to left
Multiplicative	<b>*, /, %</b>	left to right
Additive	<b>+, -</b>	Left to right
Relational Operators	<b>&lt;&gt;, &lt;=, &gt;=</b>	Left to right
Equality	<b>==, !=</b>	Left to right
Logic AND	<b>&amp;&amp;</b>	Left to right
Logic OR	<b>  </b>	Left to right
Assignment	<b>=, +=, -=, *=, /=, %=</b>	Right to left

\*\*\*\*\*

## 6.

# Function & Recursion

**Q. What is Function?**

**Ans=>** function is a block of instructions having **identity** which **takes input** it as an **argument**, process that input and produce an output. A **function declaration** tells the compiler about a function's name, return type, and parameters.

**Syntax:**

```
return_type function_identity( parameter list ) {  
    body of the function  
}
```

**>>>Parts of function:-**

**1) Return type:** A function may return a value. Return type means which type of data function will return it must be specified.

Some functions perform the desired operations without returning a value. In this case, the return\_type is **void**.

**2) Identity:** This is the actual name of the function by the help of which function Called from main function.

**3) Parameters:** parameters are placeholder for arguments passing from main function. Parameters are optional that is, a function may contain no parameters according to the Condition.

**4) Function Body:** In the body a function perform its specific task.

### **\*\*\*Function Declaration\*\*\***

A function declaration tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately. Function declaration is also called function prototype.

#### **Declaration or Prototype:**

`function_name( parameter list );`

**Ex.=>** `int max(int num1, int num2);`

### **\*\*\*Calling a function\*\*\***

Function Calling means passing value/values from main function to perform specific task. function always calling from another function like main, etc. we can pass value from main function directly or indirect.

In C programming there are 2 ways of calling function:

1. **Call by Value:** passing value directly or indirectly from the main function to the function parameter as an argument is called Call by Value. In this case, changes made to the parameter inside the function have no effect on the argument. In call by value function parameter must be variables to store that values.
2. **Call by Reference:** passing addresses of variables directly or indirectly from the main function to the function parameter as an argument is called call by reference. This means that changes made to the parameter affect the argument. We are passing address of arguments with the help of Pointers. In Call by Reference concept function parameter must be pointer to store the addresses of variables.

### **\*\*\*Type of Function in C\*\*\***

**There are two types of function in C programming:**

1. **Predefined Functions:-** C library is the collection of header file and each header file is the collection of predefined functions.

For example we use **printf()** and **scanf()** function by declaring **stdio.h** header file through **#include** Preprocessor directive.

**Some Header files and their Predefined Functions :-**

**math.h** => pow(), rand(), sqrt(), etc..

**graphics.h** => setcolor(), setbkcolor(); circle(), bar(), line(), outtextxy(), textstyle(), etc...

**string.h** => strlen(), strrev(), strcat(), strcpy(), strcmp(), etc...

**conio.h** => clrscr(), getch(), getche(), textcolor(), textbackground(), etc...

**2. User-defined functions:** You can also create functions as per your need. Such functions created by the user are known as user-defined functions. These functions can only be called from main function directly or indirectly.

### **\*\*\*why we are using functions\*\*\***

The advantages of using functions are:

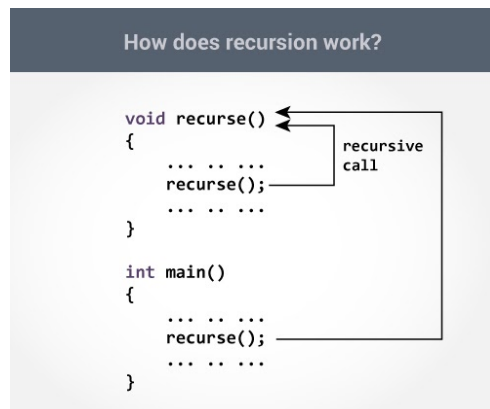
1. Avoid repetition of codes.
2. Increases program readability.
3. Divide a complex problem into simpler one.
4. Reduces chances of error.
5. Modifying a program becomes easier by using function.

### **Q. What is Recursion? Types of Recursion?**

**Ans=>** A function that calls itself is known as a recursive function. And, this technique is known as recursion.

The recursion continues until some condition is met to prevent it. If we are not giving any condition then it will work as an infinite loop. So one condition is compulsory for terminating recursive function.

## Syntax:



**Ex.=>** Factorial of a given number by recursion using this formula:  **$\text{fact}(n) = n * \text{fact}(n-1)$**

## Types of Recursions

There are 2 type of Recursion:-

- 1. Direct Recursion:** the function Calls itself again is called direct recursion. And that function is called direct recursive function. Means it is normal recursion which we discussed above.

**Example=>** Factorial of a given number by recursion using this formula:  **$\text{fact}(n) = n * \text{fact}(n-1)$**

```
#include<stdio.h>
int fact(int);
int main()
{
    int num,f;
    printf("Enter Number: ");
    scanf("%d",&num);
    f= fact(num);
}
```

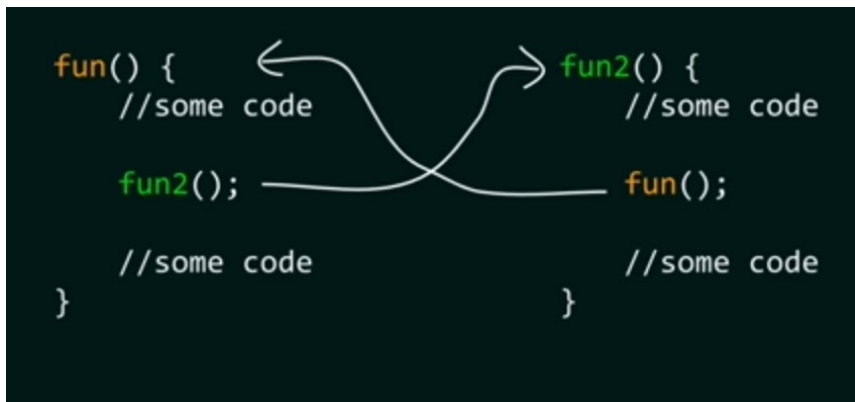
```

    printf("factorial of %d is %d",num,f);
    return 0;
}
int fact(int n) {
    if(n>=1)
        return n*fact(n-1);
    else
        return 1;
}

```

**2. Indirect Recursion:** when **2** functions calling each other in circular manner is called Indirect recursion. And that function is called Indirect recursive function

### Syntax of Indirect Recursive Function:



### Program of Indirect Recursive Function:

```

/* Program to print number from 1 to 10 in such a way that
when number is odd, add 1 and when Number is even,
subtract 1.*/
#include<stdio.h>

```

```
void odd();
void even();
int n=1;
int main() {
    odd();
    return 0;
}
void odd() {
    if(n<=10) {
        printf("%d ",n+1);
        n++;
        even();
    }
    return;
}
void even() {
    if(n<=10) {
        printf("%d ",n-1);
        n++;
        odd();
    }
    return;
}
```

\*\*\*\*\*



# Pointers

## Q. What is Pointer?

**Ans=>** pointer is variables that stores the address of another variable. Generally pointers are used to access the information of a memory location.

### Syntax:

data type\* variable;

Or

data type \*variable;

**Declaration:** int \*a;

## Type of Pointer :-

**1. Typed Pointer:** always points to specific type of data.

Like integer pointer, double pointer, struct emp pointer, string pointer, array pointer.

**2. Untyped Pointer:** Can points to any type of data. Also called generic pointer. Only void pointer comes under generic pointer.

Generally in pointer concept we need to take the help of two Operators to perform any task:

**1. Address of operator (&):** used to returns the address of a perticular variable.

Ex.=> &i=> 87994,  
&j=>87998

**2. Pointer Operator (\*):** used to return the value of address or memory location.

Ex.=> \*(&i) = 72  
\*(&j) = 87994 //it holds the address of pointer.

**program for better understanding:-**

```
void main(){
int i=100;
int *ptr;
ptr=&i;
printf("%d\n",i); // Value of i
printf("%d\n", *(&i); // Value of i
printf("%d\n", *ptr; // Value of i
printf("%u\n", ptr; // Address of i
printf("%u\n", &i; // Address of i
printf("%u\n", &ptr; // Address of Pointer
}
```

**Note:-**

- *Pointer always holds 2 byte memory because it only store address of a variable not value. Address of variable is always **unsigned integer type**. That's why we are*

*using "%u" format specifier for printing address of a variable.*

- *Pointer size always vary on compiler type or computer architecture. If it is **16 bit compiler** then the size of int and integer pointer is **2 byte**. And if it is **32 bit compiler** then the size of of int and integer pointer is **4 byte**.*

### **\*\*\*Pointer to function\*\*\***

**Calling function with the help of pointer is called Pointer to function.**

Generally we know that how to create pointer to primitive data type like float, char, int, double ect. the declaration of a pointer in all the primitive data type is same. But in the concept of function pointer declaration is totally different.

***declaration of a pointer to function is completely depend on the prototype of function.*** Means that the function prototype and declaration of pointer should be same.

**Syntax:** return type (\*identity) (arg list);

**Declaration:** int (\*ptr) (int, int)

This is the exact Declaration of pointer to function. In this example this pointer can points to any function which is taking two integer arguments and return integer data.

## How to call any function by pointer?

To call any Function anywhere in the program we need to **assign address of that Function** into pointer using Address of Operator.

**Syntax:** pointer = address of function

**Ex=>** ptr=&add

## Program for better understanding:-

```
#include <stdio.h>
int add(int, int);
int subs(int, int);

int main(){
    int (*ptr) (int,int). // This function pointer points to both function

    ptr= &subs; //calling subs function
    printf("%d\n", subs(9,5);

    ptr= &add; // calling add function
    printf("%d", add(4,3);
}

int add(int a, int b){
    return a+b;
}

int subs(int a, int b){
    return a-b;
}
```

**Output:** 4

7

**Explanation:** in above Program one pointer point to 2 function one is **subs function** and second is **add function**. Means this Pointer can points to those functions in program which is accept 2 argument and return integer data. We can call any function after assigning **address of function** into pointer using **Address of Operator (&)**.

### **\*\*\*Dangling Pointer\*\*\***

**Q. What is Dangling Pointer?**

**Ans=>** dangling pointer is a pointer which holds the non-existing memory address. Means we can say that Dangling pointer is a pointer pointing to a memory location that has been freed (or deleted).

There are different different conditions in which any pointer become dangling pointer :-

- When Variable goes out of scope
- When Function call
- When De-allocation of memory by free function

**1) Variable goes out of scope :-**

```
#include<stdio.h>

int main(){
int *ptr;
```

```
{  
int x;  
ptr=&x;  
ptr=NULL //to avoid dangling pointer problem  
}  
    return 0;  
}
```

**Explanation:** In above Program we Declare a pointer ptr which Store the address of x variable. means ptr points to x variable.

Here, ptr is become a dangling pointer after the execution of inner block in which **x variable** is Declared. because scope of **variable x** is only in inner block means **x variable** will not exist out of inner block.

now x variable will be deleted due to its scope. And allocated memory location of variable x is also freed.

Now, but ptr still hold the address which was previously freed (deleted) by program. Means ptr still Store non-existing memory address. That's why ptr is called dangling pointer of that program.

To avoid this problem we assign NULL to pointer ptr. Now ptr returns NULL pointer means 0.

## 2) When Function Call:-

The pointer pointing to local variable becomes dangling when local variable is not **static**.

when function call to main function than "**segmentation fault**" error will be show.

**Ex=>**

```
int *show(void) {  
    int n = 76; /* ... */  
    return &n;  
}
```

### **3) When De-allocation of Memory by free function**

```
main() {  
    float *p = (float *)malloc(sizeof(float));  
    //dynamic memory allocation.  
    free(p);  
    //after calling free() p becomes a dangling pointer  
    p = NULL;  
    //now p no more a dangling pointer.
```

\*\*\*\*\*

## **8.**

### **Array**

**Q. What is Array?**

**Ans.=>** An array is a variable that can store multiple values of the same data type. Array is Comes under derived data type. For example, if you want to store 100 integers, you can create an array for it.

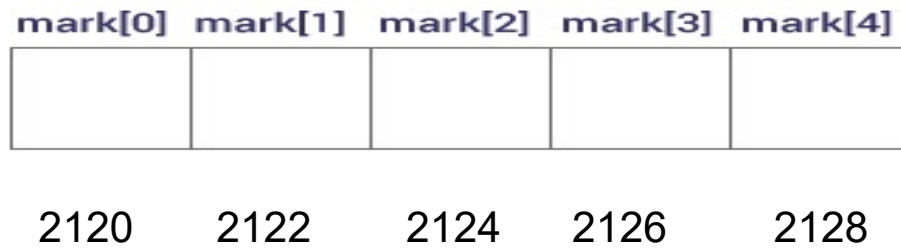
With the help of Array we can store **n number of elements** in a **single variable of same data type**.

**Syntax:-** dataType identity[size of array];

**Declaration:-** int marks[5];

**Initialisation:-** int marks[5]={12,23,34,45};

### Q. How Memory will be Allocated in Array Concept?

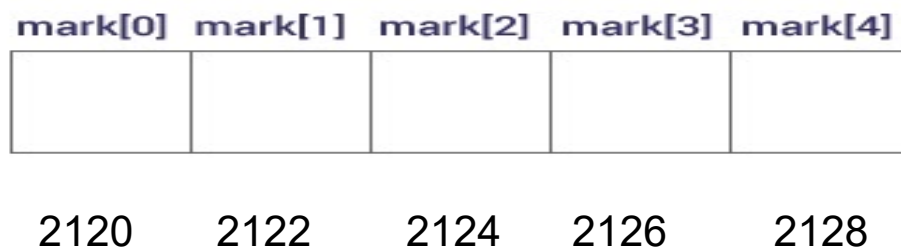


1. Suppose the address of mark[0] is 2120. Then, the address of the mark[1] will be 2122. Similarly, the address of mark[2] will be 2124 and so on. This is because the size of a int is 2 bytes. That's why the address gapping is also 2.
2. Array variable always holds the base address of memory block. Here **marks** variable holds the address of first element which is suppose 2120. That means the base address of **marks array** is 2120.
3. In **primitive data type** like int, float, char we use **pointer** for storing the address of a variable. But in array for storing address of all elements, pointer is not used. Here array variable holds the base address that is why it is called **internal pointer variable**. In above example **marks** variable is used to store address of array.



## How to Access an Array Elements?

We can access array elements with the help of indexing number like marks[0], marks[1],...marks[4]. Array Indexing Start with 0.



Here marks variable holds base address of array memory block which is called **Internal pointer variable**. Address of marks is 2120.

If the size of an array is n, to access the last element, the n-1 index is used. In this example, mark[4] is Last element.

## Q. How to initialise array?

**Ans=>** There are some common ways of initialising array:-

### 1) Direct initialisation using Curly Braces { }:

Ex=> int marks[4]={45,67,56,34};

### 2) By Assigning value one by one:

Ex=> int mark[4];

marks[0] = 45;

marks[1] = 67;

marks[2] = 56;

marks[3] = 34;

**3) By using Loop:** storing values from user by loop and also printing with the help of loop.

Ex=>

```
#include <stdio.h>
int main() {
    int arr[4];

    for(int i=0; i<4; i++) {
        printf("Enter Element %d: ", i+1);
        scanf ("%d",&arr[i]);
    }

    for(int i=0; i<4; i++) {
        printf("Value of element %d is %d\n", i+1, arr[i]);
    }
    return 0;
}
```

**Q. How to process (use) the elements of array?**

**Ans.=>** We can process the elements of Array using iterators like while loop, for loop, do-while loop.

### **\*\*\*Local & Global Declaration of an Array\*\*\***

**Local Declaration:** declaration of an array inside of any function is called local declaration of an array. There are three situation in which local declaration initialised:-

**1) Case 1:** int arr[4];

In this case, if we are not initialising array then all the elements of array initialised with **garbage values**.

**2) Case 2:** `int arr[4]={23,34};`

In this case remaining elements of array initialised with 0.

**3) Case 3:** `int arr[4]={23,34,45,12};`

In this case, all the elements of Array normal initialised with given values.

**Global Declaration:** declaration of an array out of all the functions is called Global declaration of an array. Means that in global declaration all the elements initialised with 0.

**Examples:**

- `int arr[4];` => initialised with 0.
- `int arr[4]={12};` => remaining elements initialised with 0.
- `int arr[4]={12,23,45,9};` => all the elements initialised with given values.

### **\*\*\*Types of array\*\*\***

There are three types of array in C Programming :-

**1) One Dimensional Array:** The type of array which is having only one subscript is called 1-D array. 1-D Array is used to store data in linear form.

**Syntax:-** `dataType identity[size of array];`

**Declaration of 1d array:** `int arr[4];`

**Initialisation of 1d array:** `arr[4]={23,45,67,12};`

we are initialise 1-D array just like variable. But the main difference is that we are using curly braces { } for initialising all the elements of array.

**2) Two Dimensional Array:** The type of array which is having 2 subscript is called 2-D array. Two dimensional array is also called matrix or table.

**Syntax:**

`dataType identity[Number of Rows][Number of Columns];`

There are **two methods** of declaring & initialising 2D array :-

**Method 1:**

**Declaration=>** `int arr[2][3];`

**Initialisation=>** `arr[2][3]={12,13,14,15,16,17};`

In this method we are initialising all the elements of two rows in single line

**Method 2: declaration=>** `int arr[2][3];`

**initialisation=>** `arr[2][3]={  
                  {12,13,14};  
                  {15,16,17};`

};

In this method we are initialising element of 1st and 2nd row separately using two inner curly braces. One curly brace for 1st row and 2nd curly brace for 2nd row. Both curly braces enclosed with another outer curly brace.

**3) Multidimensional array:-** The type of array which is having more than 2 subscript is called M-D array. In C programming, you can create an array of arrays is known as multidimensional arrays.

**Syntax:**

dataType identity[N of 2d array][N of Rows][N of Columns];

**Declaration:** int arr[2][3][4];

**Initialisation:** int test[2][3][4] = {  
                          {{3, 4, 2, 3}, {0, -3, 9, 11}, {23, 12, 23, 2}},  
                          {{13, 4, 56, 3}, {5, 9, 3, 5}, {3, 1, 4, 9}}  
                          };

If any array having 3 subscript then the total elements of array is [sub1]\*[sub2]\*[sub3].

**For example:** int arr[2][3][4];

This M-d array can holds  $2*3*4=24$  elements.

**Array Program example:**

```
/* Program to find smallest number in list*/  
#include<stdio.h>  
int main()  
{ int list[100];
```

```

int n, min;
printf("Enter Limit: ");
scanf("%d",&n);
for(int i=0; i<n; i++) {
    printf("Enter Number %d: ", i+1);
    scanf("%d",&list[i]);
}
min= list[0];
for(int i=1; i<n; i++) {
    if(list[i]<min) {
        min= list[i];
    }
}
printf("Smallest Number is %d",min);
return 0;
}

```

### \*\*\*Pointer to Array\*\*\*

#### Q.What is Pointer to Array?

**Ans=>** we all know that a Pointer holds the address of a variable. If pointer is int type than it holds the address of only int variable. In same way ***pointer to Array is the collection of addresses in a single pointer variable*** like int (\*ptr)[10]. In this example one pointer variable holds n number of addresses due to Array of pointers.

For example: if we want to store the addresses of array elements ( like int arr[5]) than we will create array of pointers (like int (\*ptr)[5]). Now this array of pointers is capable to hold the address of each of the element one by one in contigues block.

**Ex=>**

```
#include<stdio.h>
int main()
{
    /*pointer to single variable*/
    int a=2;
    int *p;
    p=&a;
    /*printing the values of a by pointer p*/
    printf("value of a is = %d\n", *p);

    /*Pointer to Array*/
    int arr[5]= {12,34,56,33,21};
    int (*ptr)[5]; //array of pointers
    ptr=&arr;

    /*printing values of array by for loop*/
    for(int i=0; i<5; i++) {
        printf("value of arr[%d] is = %d\n",i, (*ptr)[i]);
    }

    return 0;
}
```

Output:

```
value of a is = 2
value of arr[0] is = 12
value of arr[1] is = 34
value of arr[2] is = 56
value of arr[3] is = 33
value of arr[4] is = 21
```

**Explanation:-**

- In this program we differentiate pointer to variable and

pointer to array. The process of creating array pointers is just same like pointer to variable.

- but the only main difference is that we are creating array of pointers for storing address of Each of the element of array.
- And one more difference is that the syntax of pointer to array is different like that `int (*ptr)[5]`. Means we use paranthesis for pointer to array and after that giving size like `[5]` means it can hold 5 addresses.

\*\*\*\*\*

## 9. String

### Q. What is String?

**Ans=>** string is a one dimensional character array terminated by a null (`'\0'`). When the compiler encounters a sequence of characters enclosed in the single quotation marks, it appends a null character `\0` at the end by default. Null Character (`\0`) Means terminating String and the ASCII value of Null (`\0`) character is 0.

**Syntax:-** `char identity[size];`

**Declaration:-** `char name[20];`

### Q. How Memory Will be Allocated in String?

c[0]	c[1]	c[2]	c[3]	c[4]
a	b	c	d	\0
2000	2001	2002	2003	2004



- In Sting variable (array name) holds the base address of string memory block. string variable is also called Internal pointer variable.
- Each of the character of string takes 1 byte memory for Allocation.
- In above example there are 4 characters in c string but including null Character (\0) there are 5 Characters.
- Suppose the address of c[0] is 2000. Then, the address of the c[1] will be 2001. Similarly, the address of c[2] will be 2002 and so on. This is because the size of char is 1 bytes. That's why the address gapping is also one.

### Q. How to initialised a string?

You can initialize strings in number of ways:-

1. `char c[] = {'a', 'b', 'c', 'd', '\0'};`
2. `char c[5] = {'a', 'b', 'c', 'd', '\0'};`
3. `char c[] = "abcd";` // Value after d is '\0' by default
4. `char c[50] = "abcd";` // Value after d is '\0' by default

### Q. How to access the Elements (characters) of string?

**Ans=>** we can access the characters of string through indexing like `name[0]`, `name[1]`, `name[2]`... and so on. String Indexing Start with 0.

The last element of string is null character which is **automatically added** when we are initialising string in such way:-

**`char str[20]="abcd";`**

## Q. How to process the elements of string?

Ans=>

- In string we use **%s** format specifier for reading & writing all the character of string at once.
- For **%s** there is no need of iterators (loops).
- We can also use **%c** format specifier of character but through this format specifier we can access one character at a time. For accessing all the elements of string or word we use **%S**.
- There is no need to provide address when we are using **%s** format specifier, because variable of string holds the Base address of string and from the base address what all characters End-user input character by character **%S** will collect by scanf function and print by printf function.

### \*\*\*String Handling Functions\*\*\*

Some Special Functions iof string.h header file are used for handed any string in program.

These Special Functions are as follows:-

1. **gets()** => The gets() function in C reads characters from stdin (standard input) and stores them until a newline character is found or end of file occurs. When gets funtion encounter new line character then it will terminate and stop reading characters.

In Simple words we can say that this Function is used to read multi-strings until new line character. When we press enter than it will be terminate. scanf function can only read one string thats why we are using gets() function to read multi-strings like line, paragraph.

**Syntax:** char \*gets(char \*str);

**Declaration:** gets(str);

**Parameter:** Here, in the parameter of this function **str is a pointer to an array of chars** (ArrayName holds the base address) where the String is stored.

**Return:** this function returns str on success, and on error it will return NULL.

**Example:**

```
#include <stdio.h>

int main () {
    char str[50];

    printf("Enter a string : ");
    gets(str);

    printf("You entered: %s", str);

    return(0);
}
```

**2. puts() =>** The puts() function in C print a string to stdout (Standard Output) but not including the null character. A newline character is appended to the output.

In simple word we can say that it is a Function which is used to print multi-strings on console. printf function can only print one string that's why we are using puts Function for printing multi-string like line, paragraph.

**Syntax:** int puts(const char\* str);

**Declaration:** puts(str);

**Parameter:** Here in the parameter of this function str is the constant string that is to be printed.

**Return Value:** If successful, non-negative value is returned. On error, the function returns EOF.

### Example:

```
#include<stdio.h>
#include<string.h>
int main()
{char name[20];
printf("Enter your name: ");
gets(name);
printf("Your name is ");
puts(name);
    return 0;
}
```

**3. strlen()** => this function of string.h header file is used to calculate the length of string excluding Null ('\0') character.

The strlen() function takes a string as an argument and returns its length. The returned value is of type size\_t (the unsigned integer type).

**Syntax:**

size\_t strlen( const char\* str)

**Declaration:** strlen(str);

**Example:**

```
#include<stdio.h>
#include<string.h>
void main() {
    char name[]="Dheeraj Patidar";
    printf("Length of Your name is: %d", strlen(name));
}
```

Output: Length of Your name is: 15

**4. strcat()** => this function of string.h header file is used to concatenates (joins) two string. This Function takes two arguments. first is destination string and 2nd is Source string.

The strcat() function concatenates the destination string and the source string, and the result is stored in the destination string.

**Syntax:**

char \*strcat(char \*destination, const char \*source);

### Example:

```
#include<stdio.h>
#include<string.h>
int main()
{char str1="Hello ";
 char str2="World.";
strcat(str1,str2);
printf("Concatenation string is = %s", str1);
 return 0;
}
```

Output: Concatenation String is = Hello World.

**5) strcpy() =>** this function of string.h header file is used to copy one string into another including Null character ('\0').

**Syntax:** char\* strcpy(char\* destination, const char\* source);

**Declaration:** strcpy(str1,str2);

when we are using this function, the destination size should be equal or greater than source string. Otherwise this function misbehave.

### Example:

```
#include<stdio.h>
#include<string.h>
int main()
{ char str1="Hello";
 char str2="World";
printf("value of str1 is = %s",str1);
```

```
printf("value of str2 is = %s",str2);
```

```
strcpy(str1,str2);
```

```
printf("Now value of str1 is = %s", str1);
```

```
return 0;
```

```
}
```

Output: value of str1 is = Hello  
value of str2 is = World  
Now value of str1 is = World

**6) strcmp** => this function of string.h header file is used to Compare 2 string and return integer value.

**Syntax:** int strcmp(const char \*str1, const char \*str2);

**Declaration:** strcmp(str1,str2);

### **Some ASCII System Value Points:-**

- The integer values of all upper case letters (65-90) are less than integer Values of all lower case letter (97-122).
- Digits value are less than letters values (48-57).
- Space character ASCII Value (32) is less than all the printed characters.

**String Comparison function** returns integer value in such a way that:-

- If  $s1 < s2$ , then it returns a value which is less than 0.
- If  $s1 > s2$ , then it returns a value which is greater than 0.
- If  $s1 == s2$ , then it returns 0.

**How strcmp function work:-**

- strcmp function first compare the length of 2 strings. According to condition it will give output but if the length of both strings are equal than it compare one by one character of both strings.
- Whenever it found any mismatch character than it will Compare both characters ASCII Value and according to condition it return value.
- Now if all the characters of both strings are same than it will return 0.

### Example:

```
#include<stdio.h>
#include<string.h>
int main()
{  char str1[10];
   char str2[10];
   int cmp;

   strcpy(str1,"AbcD");
   strcpy(str2,"Abcd");

   cmp=strcmp(str1,str2);
   if(cmp<0) {
       printf("str1 is less than str 2\n");
   }
   else if(cmp>0) {
       printf("str2 is less than str1\n");
   }
   else {
       printf("both are equal\n");
   }
}
```



```
}  
printf("the value return by strcmp is: %d",cmp);  
return 0;  
}
```

Output: str1 is less than str 2  
the value return by strcmp is: -128

\*\*\*\*\*

# Structure

## Q. What is Structure?

**Ans=>** structure is a user defined data type. Using structure we can Store **n** number of elements of **different data type** in a single variable.

For example in **Array Concept** we can store **n** number of element but of **same data type**. due to this disadvantage of array the concept of structure is introduced.

### Syntax:

```
struct structureName
{
    dataType member1;
    dataType member2;
    .....
    .....
};
```

### Declaration:

```
struct emp
{
    char ename[20];    //use of string for storing name
    int ecode;
    float esalary;
};
```

One more important thing about structure is that Primitive data type having only single word like int, float, char. but user Defined dataType is a collection of word like structure emp.

**Ex=>**

```
#include<stdio.h>

struct emp
{
char ename[20];
int ecode;
float esalary;
};

int main(){
int a; // Here int is a dataType of variable a
struct emp e1; // here struct emp is a dataType of variable e1
/*

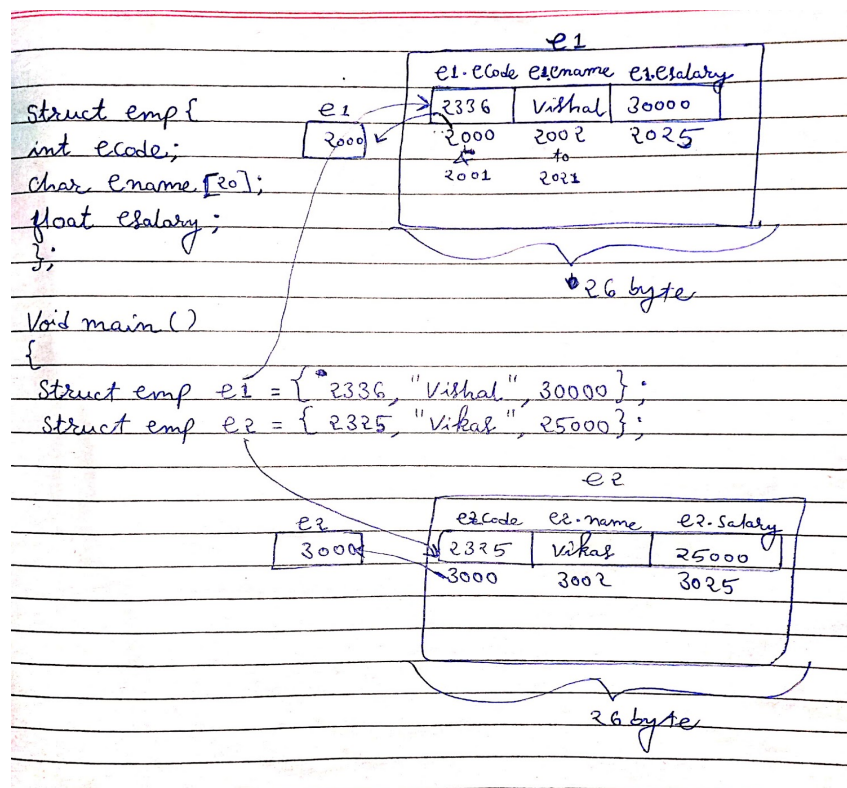
.....
some code

.....
*/
}
```

**Q.How Memory will be allocated in structure?**

**Ans=>**

- When a struct type (struct emp) is declared, no storage or memory is allocated. To allocate memory of a given struct type (struct emp) and work with it, we need to create variables (Like e1, e2, etc).
- When we are Declaration variable (e1), a block of memory will be allocated for (e1) variable according to structure members (like ename, ecode, esalary).
- ename get 20 byte, ecode get 2 byte, esalary get 4 byte in 16 bit Compiler. Means total memory will be allocated for (e1) variable is total of all members (20+2+4=26) byte.
- **How much memory will be allocated by structure variable is totally depend on the members of structure.**



- In this diagram, e1 holds 26 byte memory and also e2 holds 26 byte memory. for both e1 and e2 variables separate memory block will be allocated. Inside of these blocks 3 variables declared due to structure. e1 holds 2000 to 2025

memory addressed and e2 holds 3000 to 3025 memory addresses. here e1 is called Internal pointer variable of its memory block and e2 is also called Internal pointer variable of its memory block because they store's the base address of their Memory block.

### **Q. How to access the members of structure?**

**Ans=>** There are two types of operators used for accessing members of a structure.

- **Accessor (.) =>** Used in structure to access the members of structure by variables.  
For example: e1.ename, e1.ecode, e1.esalary, etc.
- **Arrow operator (->) =>** used in the concept of **pointer to structure**.  
for example: ptr->ename, ptr->ecode, ptr->salary, etc.

### **Q. How to initialise structure?**

**Ans=>** we can initialise any structure just like an array. but the main difference b/w initialisation of structure and array is that there is no need to specify the size of structure variable because we are using the member of a structure for storing elements. And in the concept of array it is mandatory to specify the size of array because all the elements are of same data type.

**For Example:**

```
struct emp e1={"vishal",6253, 30000};  
int arr[10]={12,34,56,22};
```

There are 2 ways of initialising structure:-

- 1) Direct Initialisation.
- 2) Input values from End-user using printf function.

### **\*\*\*Size of Structure by sizeof Operator\*\*\***

we can find sizeof structure withb2 method:-

#### **1) By passing structure variable:**

```
printf("Size of Structure is %d", sizeof(e1));
```

#### **2) By passing struct type:**

```
printf("Size of Structure is %d", sizeof(struct emp));
```

### **\*\*\*Local & Global Structure\*\*\***

**Local structure:** A Structure which is declared inside a particular function is called local structure.

and the declaration of a structure inside the function is called local declaration of structure.

**Global structure:** A structure which is declared outside to all the functions is called Global structure.

And the declaration of a structure outside to all the functions is called Global declaration of a structure.

### \*\*\*Pointer to Structure\*\*\*

we can also use pointer concept in structure. In this concept we are just creating struct type pointer (like struct emp \*ptr;) and assign the **address of variable of a structure** to that Pointer (like struct emp \*ptr=&e1;).

Now we can access the members of structure through pointer by passing address of variable. In this concept we can access members of struct type by **arrow operator** (->) in place of **accessor** (.).

#### Example:-

```
#include<stdio.h>
struct emp{
char name[20];
int code;
float salary;
};
int main()
{
struct emp e1; //Declaring a Variable
struct emp* ptr; //Declaration of structure pointer
ptr= &e1; //assigning address of e1 variable to ptr
printf("Enter name of e1: ");
scanf("%s",ptr->name);
printf("Enter code of e1: ");
scanf("%d",&ptr->code);
printf("Enter salary of e1: ");
scanf("%f",&ptr->salary);

printf("\nname of e1 is: %s\n", ptr->name);
```

```

printf("code of e1 is: %d\n",ptr->code);
printf("salary of e2 is: %.2f\n",ptr->salary);
return 0;
}

```

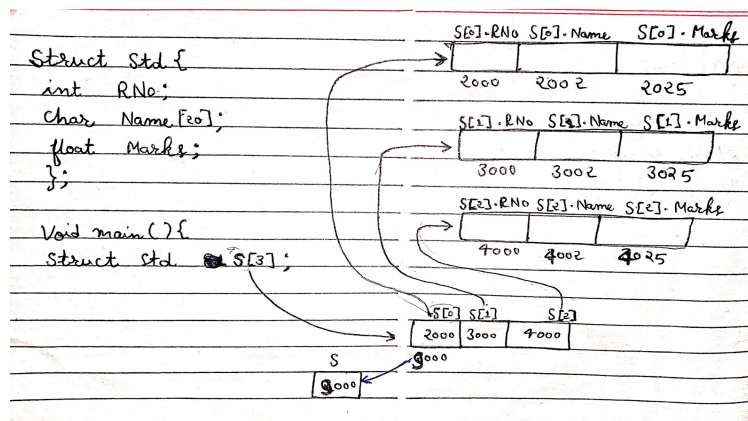
### \*\*\*Array of structure\*\*\*

As we all know that structure is a collection of different data types. Whereas array of structure is nothing but the collection of structure.

For example if we want to store the record of one student by structure but if we want to store the records of n number of students then it becomes very complicated and waste of time, that's why we are using the concept of array of structure. In this concept we can collect a number of Students records.

In the Concept of Array of structure all the syntax is same as simple structure. Only Multiple structures are used.

### More Clarity With example:-



\*\*\*\*\*



# 11.

## Union

**Q. What is union in c programming?**

**Ans=>** union is a user defined data type like structure. But union members share same memory location for storing and processing the data.

**Example=>**

```
Example:

struct abc {
    int a;
    char b;
};

union abc {
    int a;
    char b;
};

a's address = 6295624
b's address = 6295628

a's address = 6295616
b's address = 6295616
↓
```

In above example we found that in structure a and b both members get different memory location but in union a and b share same memory location.

**Syntax:**

```
union unionName
{
    dataType member1;
    dataType member2;
    .....
    .....
```

```
};
```

### Declaration:

```
union abc
```

```
{
```

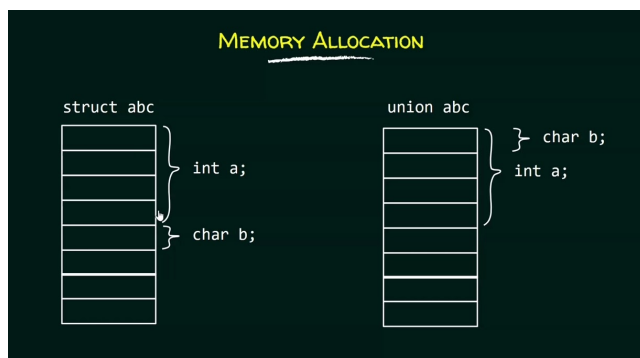
```
int a;
```

```
char b;
```

```
};
```

### Q. How Memory Will be allocated in Union Concept?

Ans=>



- In above example, structure int a get 4 byte memory and after that char b get 1 byte memory means both members get different memory.
- But in union both int a and char b are sharing the same memory location and that is why both get same memory address.
- In union, members will share same memory location and if we change value of one member than the values of another variable will replaced with that values. for example int a=10 and int b=20, then value of a become 20 due to replacement of value.

**Example for more clarity:-**

```

union abc {
    int a;
    char b;
}var;

int main() {
    var.a = 65;
    printf("a = %d\n", var.a);
    printf("b = %c", var.b);
    return 0;
}

```

OUTPUT:

a = 65  
b = A

1. In this above example, we declare a union of member int a, and member char b. Now after we declare a variable var.
2. Through accessor (.) We initialise **var.a** by 65 and after that print it.
3. Now we will print value of b member by var.b.
4. Now in output we found that value of a is 65 and value of b is also 65 but it will converted into char thats why we found A in output.

### \*\*\*Size of Union\*\*\*

Size of union is depend on the size of the largest member of the union.

**Example for more Clarity:-**

Example:

```

union abc {
    int a;
    char b;
    double c;
    float d;
};

int main() {
    printf("%ld", sizeof(union abc));
    return 0;
}

```

OUTPUT: 8

In this above example from all the members of abc union, double c is largest in size. Because we all know that double data type is of 8 byte in any system and char, float, int having smaller size

compare to double Data type. That's way the size of union abc is 8 byte.

As we all know that `sizeof()` operator is used to find the size of any primitive, derived, and user-defined data type. We can use either union type (`union abc`) or variable of union (`var`) for finding size of union.

### Q. How to Access the members of Union?

**Ans=>** Accessing the members of Union is same as structure.

But we are discussing it again for better understanding..

There are two types of operators used for accessing members of a union.

- **Accessor (.) =>** Used in union to access the members of union by variables.

For example: `e1.ename`, `e1.ecode`, `e1.esalary`, etc.


- **Arrow operator (->) =>** used in the concept of **pointer to union**.

for example: `ptr->a`, `ptr->b`, `ptr->c`, etc.

### Example of Arrow operator in pointer to union concept:-

```
union abc {
    int a;
    char b;
};

int main() {
    union abc var;
    var.a = 90;
    union abc *p = &var;
    printf("%d %c", p->a, p->b);
    return 0;
}
```



\*\*\*\*\*

# 12.

## Dynamic Memory Allocation

### Q. What is Static Memory Allocation?

**Ans=>** Fixed Memory allocated during the run time is called static memory. We called static memory is a fixed memory because this memory size is decided by programmer. If user cannot Store any element in array then wastage of memory is occur and if user want to input element across the limit then it is not possible. That's why we called static memory is a fixed memory.

### Some Problems in Static Memory Allocation:-

- 1) If you want to allocate memory for an array during run time then you have to fix the size of array at the time of declaration because Size is fixed and user cannot increase or decrease the size at run time.
- 2) If the values stored by the user in array is less than the specified size then there will be wastage of memory occur.
- 3) If the values stored by the user in array is more than the specified size then the program may crush or misbehave.

### Q. What is dynamic memory allocation?

**Ans=>** To avoid all the problems during static memory they introduced **dynamic memory allocation** concept. When Memory allocation is depend on the increment and decrement of the data is called **dynamic memory allocation**.

We can say that Sometimes the size of the array you declared may be insufficient. To solve this issue, you can allocate memory

manually during run-time. This is known as dynamic memory allocation in C programming.

### **Q. How to allocate memory Dynamically?**

**Ans=>** To allocate memory dynamically following 4 function of stdlib.h Header file are used:-

#### **1) malloc ( ) :-**

- ~~used to allocate memory to structures dynamically.~~ The name "malloc" stands for memory allocation.
- The malloc() function allocate a block of memory of the given number of bytes. And, it returns an address of a memory block on success in the form of void pointer which can be casted into pointers of any form like int\*, float\*, char\*, struct emp\*, etc.
- malloc taking only one argument which is size (byte).
- malloc function create a block in Memory and pass address of this block, which is need to be store in a pointer. Now we are handled all the data by pointer only.
- All value in malloc initialised with garbage values.
- it returns NULL Pointer on failure.
- **Syntax:** void \*malloc(int num);  
**or**  
ptr = (castType\*) malloc(size);
- **Declaration:** ptr = (float\*) malloc(100 \* sizeof(float));

#### **2) calloc ( ) :-**

- ~~used to allocate memory to array dynamically.~~ The name "calloc" stands for contiguous allocation..
- And, it returns an address of a memory block on success in the form of void pointer which can be casted into pointers of any form like int\*, float\*, char\*, struct emp\*, etc.
- calloc taking 2 arguments, one is number of elements, and second is size of the each element.
- Calloc function create n number of block According to end-user and pass base address which is need to be store in a pointer.
- All value in calloc initialised with zero (0).
- it returns NULL Pointer on failure.
- **Syntax:** void \*calloc(int num, int size);  

**or**

ptr = (castType\*)calloc(n, size);
- **Declaration:** ptr = (int\*) calloc(25, sizeof(int));

### 3) realloc ( ) :-

- used to resize memory which is previously allocated by malloc or calloc function. The name "realloc" stands for **reallocation**.
- If the dynamically allocated memory is **insufficient** or more than required, you can **change the size of previously allocated memory** using the realloc() function.
- **Syntax:** void \*realloc(void \*address, int newsize);  

**Or**

ptr = realloc(ptr, m);

- **Declaration in Case of:**  
**malloc** => ptr = realloc(ptr, n2 \* sizeof(int));  
**calloc** => ptr= realloc(ptr,sizeof(int));

#### 4) free ( ) :-

- Dynamically allocated memory created with either calloc() or malloc() does not freed automatically thats why we are using free() function to release memory manually.
- It is just taking address of s block we want to free indirectly it ist taking address by pointer.
- It is not return any value.
- **Syntax:** void free(void \*ptr);
- **Declaration:** free(ptr);

#### Example 1: malloc() & free()

```
// Program to calculate the sum of n numbers entered by the
user

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

int main()
{
    int n, i, *ptr, sum = 0;

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



```
ptr = (int*) malloc(n * sizeof(int));

// if memory cannot be allocated
if(ptr == NULL)
{
    printf("Error! memory not allocated.");
}

for(i = 0; i < n; ++i)
{
    printf("Enter element %d: ", i+1);
    scanf("%d", ptr + i);
    sum += *(ptr + i);
}

printf("Sum is = %d", sum);

// deallocating the memory
free(ptr);

return 0;
}
```

### **Example 2: calloc() & free()**

```
// Program to calculate the sum of n numbers entered by the
user

#include <stdio.h>
```

```
#include <stdlib.h>

int main()
{
    int n, i, *ptr, sum = 0;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    ptr = (int*) calloc(n, sizeof(int));
    if(ptr == NULL)
    {
        printf("Error! memory not allocated.");
        exit(0);
    }

    printf("Enter elements: ");
    for(i = 0; i < n; ++i)
    {
        scanf("%d", ptr + i);
        sum += *(ptr + i);
    }

    printf("Sum = %d", sum);
    free(ptr);
    return 0;
}
```

### Example 3: realloc()

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

```

#include <stdlib.h>
int main()
{
    int *ptr, i , n1, n2;
    printf("Enter size: ");
    scanf("%d", &n1);

    ptr = (int*) malloc(n1 * sizeof(int));

    printf("Addresses of previously allocated memory: ");
    for(i = 0; i < n1; ++i)
        printf("%u\n", ptr + i);

    printf("\nEnter the new size: ");
    scanf("%d", &n2);

    // relocating the memory
    ptr = realloc(ptr, n2 * sizeof(int));

    printf("Addresses of newly allocated memory: ");
    for(i = 0; i < n2; ++i)
        printf("%u\n", ptr + i);

    free(ptr);

    return 0;
}

```

### \*\*\*Difference between Static & Dynamic Memory\*\*\*

Static Memory	Dynamic Memory
1. Static Memory is	1. Dynamic Memory is also

<p>Allocated during the run time of Program but it is a fixed size memory which is previously decided by programmer.</p> <ol style="list-style-type: none"> <li>2. we cannot increase the size of array &amp; structure at the run time of program.</li> <li>3. If the values stored by the user in array is more than the specified size then the program may crush or misbehave.</li> <li>4. there is no pointer concept is used to allocate static memory.</li> <li>5. static memory is a default memory which is allocated by computer when we declare any array or structure.</li> </ol>	<p>allocated during the run time of a program but the size of memory is depend on the increment &amp; decrement of elements.</p> <ol style="list-style-type: none"> <li>2. It is possible to increase the size of array, structure at the run time of Program.</li> <li>3. This problem is not occur in dynamic memory allocation because the size of memory is depend on the number of elements.</li> <li>4. In the concept of dynamic memory allocation pointer concept is used. Because malloc and calloc function returns the address of memory block which is created by them.</li> <li>5. Dynamic memory is a custom memory which is are located by computer but declared by malloc and calloc function according to the End-user requirement.</li> </ol>
---	---

6. No special functions are used to allocate static memory.	6. To allocate memory dynamically we need to use 4 special functions of stdlib.h header file which is malloc(), calloc(), realloc(), free().
---	--

\*\*\*\*\*

## **13.**

### **FILE I/O**

Before going to File I/O concept we discuss some important topics about data and data persistence.

## **Q. What is data?**

**Ans=>** Data is everything which we Process thought any program. we can input any type of data in program and hold these data through variables until the program produce Output. And output is also a data which is produced by any program after processing.

For example i create a program to add two numbers. After that i'm putting two values (2,4) in variables (a,b) and after that program produce desired output (6) through logic ( $c=a+b$ ) Writen in program. So in this program 2 and 4 are input data and 6 is the data produced by program.

But after running program all the data which is inputed in program is destroy or deleted because whatever the data we passed to program was loaded inside the RAM. And we all know that RAM is a volatile memory in which data is destroyed after running any program. *"we can save program in harddisk or any other memory for using again but we have no option to save the data which is receive and produced by program inside the harddisk."*

## **Q. What is data persistence?**

**Ans=>**

- ***data persistence means how as long as data is stored into the computer.***

- When we run any program by giving value and producing output in that condition there is **no persistence of data** because all the data and logics of program is loaded in **RAM**.
- If we want to save the data inside the computer then the only way is to create file. After creating the file we can Store the data inside the hard disk through files.

### **Q. What is file & uses of file?**

**Ans=>** *Files is the collection of data in which we can Store data permanently inside the hard disk.* In simple words we can that A file is a container in computer storage devices used for storing data.

Technically we can say that File is a collection of bytes that is stored on secondary storage devices like disk.

There are two types of files:

- Text files(.txt, .c) => Written in plain text.
- Binary files(.jpg, .dat) => written in 0's and 1's.

- We use file for following Purpose:-

1. When a program is terminated after once running, the entire data is lost. Storing in a file will preserve your data even if the program terminates.
2. If you have to enter a large number of data, it will take a lot of time to enter them all. However, if you have a file containing all the data, you can

easily access the contents of the file using a few commands in C.

3. You can easily move your data from one computer to another without any changes though files.

### **Q. What is File Handling or File I/O?**

**Ans=>** file handling means all the activities which are performed by a program is stored into a file and accessing that information at the time of requirement is called file handling or FILE I/O.

In simple words we can say that storing the information into the file and accessing that information at the time of requirement through some functions is called file handling.

### **Q. What is FILE Pointer?**

**Ans=>** If we want to work with file we need to use **file pointer** through which we can hold the base address of file and access it into RAM. After that we can perform some operations like opening, reading, writing, etc. Basically file pointer is a communication way b/w program & file.

**Syntax of file pointer:** FILE \*<identity>;

**Declaration:** FILE \*fptr;



### \*\*\*Basic File Operations In C Programming\*\*\*

There are 4 basic operations that can be performed on any files in C programming language. They are,

1. Opening/Creating a file
2. Closing a file
3. Reading a file
4. Writing in a file

Stdio.h header file functions are used in File Handling

#### 1. Opening/Creating a File:-

**fopen()** => this is the most important function which is used to open a file in different modes like write mode, read mode, append mode, etc. Return type of this function is FILE pointer.

**Syntax:** FILE\* fopen("file name","mode");

**Declaration:** fptr=fopen ("example.txt","r");;

Here fopen function is taking two arguments, one is **file name** (like "file1.txt") and second is in which mode you want to open file like read, write, etc.

On success, it open the file & return the address of file.

On failure, it returns NULL Pointer.

#### Program for better understanding:-

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

```
int main () {  
    FILE *fp;
```

```

    fp = fopen("file.txt", "w");

    /* .....
    .....
    ..... */

    fclose(fp);

    return(0);
}

```

### \*\*\*Opening Modes in FILE I/O\*\*\*

Mode	Meaning of mode	During Inexistence of file
<b>r</b>	Open for reading.	If the file does not exist, fopen() returns NULL.
<b>rb</b>	Open for reading in binary mode.	If the file does not exist, fopen() returns NULL.
<b>w</b>	Open for writing.	<p>If the file exists, its contents are overwritten.</p> <p>If the file does not exist, it will be created.</p>

<b>wb</b>	Open for writing in binary mode.	<p>If the file exists, its contents are overwritten.</p> <p>If the file does not exist, it will be created.</p>
<b>a</b>	<p>Open for append.</p> <p>Means Data is added to the end of the file.</p>	If the file does not exist, it will be created.
<b>ab</b>	<p>Open for append in binary mode.</p> <p>Means is added to the end of the file.</p>	If the file does not exist, it will be created.
<b>r+</b>	Open for both reading and writing.	If the file does not exist, fopen() returns NULL.
<b>rb+</b>	Open for both reading and writing in binary mode.	If the file does not exist, fopen() returns NULL.
<b>w+</b>	Open for both reading and writing.	<p>If the file exists, its contents are overwritten.</p> <p>If the file does not exist, it will be created.</p>
<b>wb+</b>	Open for both reading and writing in binary mode	If the file exists, its contents are

		overwritten.  If the file does not exist, it will be created.
<b>a+</b>	Open for both reading and appending.	If the file does not exist, it will be created.
<b>ab+</b>	Open for both reading and appending in binary mode.	If the file does not exist, it will be created
<b>There are total 12 modes of open file.</b>		

## 2. Closing a File:-

**fclose ()** => fclose() function is used to closes the opening file.

**Syntax:** int fclose(FILE \*fptr);

**Declaration:** fclose(fp);

It is taking one argument which is file pointer pointing to that file.

This function returns zero if the file is successfully closed.  
On failure, EOF is returned.

## 3. Reading a file:-

For reading file 2 Predefine functions are used:-

- 1) **fgetc()** => this function is used to read file character by character. It is taking one character at time from file. We

use loop for Collecting data by fgetc and it will stop Collecting characters when it reaches EOF (End of the file). EOF returns -1.

**Syntax:** int fgetc(FILE \*fptr);

**Declaration:** fgetc(fptr);

It is taking one argument which is file pointer pointing to that file.

2) **fscanf()** => this function is used to read number from file.

**Syntax:** fscanf(file ptr, "format specifier", var. Address);

**Declaration:** fscanf(fptr, "%d",&num);

This function is taking 3 arguments, first is file pointer, second is formate specifiers, 3rd is address of a variable.

#### 4. Writing a file:-

For Writing file we use 2 functions:-

1) **fprintf()** => this function is used to write data into any file.

**Syntax:** fprintf(file pointer, "formate specifiers", variables);

**Declaration:** fprintf(fptr, "Number is %d", num);

2) **fputc()** => this function is used to write character by character into file.

**Syntax:** fputc("character",fptr);

**Declaration:** fputc("T.", fptr);

**Note:-** There are n number of predefined functions are used in file handling concept. All above Functions are most useful.

\*\*\*\*\*

## 14. typedef

**Q. What is typedef?**

**Ans=>**

- In C programming, typedef is a keyword which is used to give a name to any dataType in a program. In simple word we can say that typedef keyword is used to create duplicate name or Alias name of any data type.
- typedef is a user defined data type.
- With the help of typedef keyword we can give duplicate name to any primitive (int, float, char), derived (array, string) , and as well as user defined datatype (structure, union).
- We can use typedef either globally and locally.
- **Syntax:** typedef dataType name;
- **Declaration:** typedef int myint;

### Example of typedef in primitive data type:-

```
#include<stdio.h>

int main()
{
/* Local Declaration of typedef in primitive datatype*/
typedef int myint; //typedef in int
typedef char mychar; //typedef in char
typedef float myfloat; //typedef in float

/* use of these duplicate names*/
myint i= 40; //in place of int we can use myint
myfloat f=123.23; //in place of float we can use myfloat
mychar c='T'; //in place of char we can use mychar

printf("value of i is %d\n",i);
printf("value of f is %.2f\n",f);
printf("value of c is %c\n",c);

    return 0;
}
```

#### **Output:**

```
value of i is 40
value of f is 123.23
value of c is T
```

### Example of typedef in derived dataType:-

```
#include<stdio.h>

int main()
```

```

{
/* typedef Declaration of array */
typedef int arr[5];

/* use of duplicate name of array */
arr x={10,20,30,40,50};

for(int i=0; i<5; i++){
printf("%d ",x[i]);
}
    return 0;
}

```

**Output:**

10 20 30 40 50

**Example of typedef in User defied dataType:-**

```

#include<stdio.h>

typedef struct /*optional name*/{
int code;
char name[20];
float sal;
}emp;

int main()
{
/* use of this duplicate name*/
emp e1={543, "Vinay", 60500};

printf("code of e1 is %d\n", e1.code);
printf("Name of e1 is %s\n", e1.name);
}

```



```
printf("salary of e1 is %.2f", e1.sal);  
    return 0;  
}
```

**Output:**

code of e1 is 543  
Name of e1 is Vinay  
salary of e1 is 60500.00

\*\*\*\*\*

## 15. Storage Classes in C

Every Variable in C Programming has two properties one is **dataType** and second is **Storage class**.

- We all know that **dataType** is a representation of data like how much memory is required, which type of data is allowed, etc. Now one more question is arise:-

**Q.** What represents **Storage class** about Variable?

**storage class** represent many things that is:-

1. Where variable gets memory allocation.
2. What is the default value of a variable.
3. What is the scope of a variable like block scope, method or function scope, program scope.
4. What is the lifetime of a variable

**>>>Types of scope:-**

**1) Block scope:-** when we declare a variable inside of any particular block then we cannot access that variable out of that block is called block scope of that variable.

**2) Method or Function scope:-** when we declare a variable inside of any particular function then we cannot access that variable in another function is called function scope of that variable.

**3) Program Scope:-** when we declare any Global variable then it can be access in any function of that program. It is called the Program scope of a variable.

## >>>Types of storage classes in C

**There are 4 types of storage classes:-**

### **1. automatic storage class:-**

In the declaration of a local variable, if we are not specifying any storage class then by default it is an auto storage class.

**Keyword:** auto

**Memory:** RAM

**Default Value:** Garbage value

**Life time & Scope:** Declaration either inside block or method. Auto variables are always local variables. We cannot Declare any auto variable globally. we can Access only within the block or method in which it define. **Lifetime** of auto variable is unti the block or method execution completes.

**Example:-**

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

```
int main(){
auto int a=10; //method scope
{
auto int a; //block scope
printf("value of a is %d\n",a); //Garbage Value
}
printf("value of a is %d",a); //10
return 0;
}
```

**Output:**

value of a is 1970428340  
value of a is 10

## 2. Register storage class :-

register variable can only be Declared locally mean with in the block or function.

**Keyword:** register

**Memory:** CPU Register

**Default value:** Garbage Values

**Life time & Scope:** declaration of Register variable is within the block or method means we can access register variable within the block or method. We cannot access and declare register variable globally. **Lifetime** of Register variable until the block or method execution completes.

- Register variable and auto variable having same logic but the main difference between them is that we can access register type variable much faster than auto variable

because auto variable gets memory allocation inside the RAM but register variable get memory allocation in the process space (part) of CPU called register.

- if you want to use any variable repeatedly like in loop concept then we should go for register variable because they are faster than auto variable.
- Every program processing is always taken place in register memory of CPU which is the fastest memory of computer. Whenever we declare any auto variable then it will allocate memory inside the RAM but RAM is not process data for this work it call to register memory inside the CPU. It should be noted that memory allocation of a program inside the RAM but 4 process RAM call to register means RAM giving one by one instruction to register and this process is time consuming.
- For this drawback of auto variable we should go for register variable. After declaring register variable in place of auto variable it will gets memory allocation directly in register memory of CPU. For register memory it is easy to process the data without taking instructions one-by-one from RAM. That is the reason auto and register variable is different from each other.

**Note:-**

- *Maximum Size of register memory is 8 to 16 bytes thats why we are not Declaring register variable globally. Local variable are temporary variables thats why they get less memory compare to global variables.*
- *It is not possible to declare all the variables with register class because register memory is maximum 16 byte in*

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

### **3. static storage class:-**

- When we declare any static variable it will not be destroyed when control comes out of the block or method. Static variable is not destroyed after the execution of any function or block; it will persist until the program execution is complete.
- In another word we can say that auto and register only work with the function and deallocate after control comes out of the function or block. If we do not want to deallocate memory of a variable then we should go for static variable.
- The keyword **static** instructs the compiler to keep a local variable until the execution of a program.
- When we declare a static variable globally it always updates its value until the program is executed.
- Static variable will not get memory allocation again and again. This type of variables get memory allocation once in a program. We can use them by updating values only.

### **Example:-**

```
#include<stdio.h>
void fun(void);
```

```
void main()
{
fun();
fun();
fun();
}

void fun(void){
static int a=5;
printf("a: %d\n", a+=2);
}
```

**Output:**

a: 7  
a: 9  
a: 11

#### 4. External Storage class:

- Extern variable is used to Declare a global variable which is already define somewhere. Here defination & Declaration both are totally different.
- for example **int a** is declaration with definition. And **external int a** is only Declaration.
- **extern** keyword is used to use any global variable of either same program or another file program into any function.
- we can say that **extern keyword** is used to import the value of any **global variable** inside any block or method which is previously defined.

- **extern** keyword is mainly used to call data in particular file which is present in another file. Means that this extern variable concept is used to share data between C project files.
- We can only reuse the data of global variable somewhere by using extern keyword.

### **Use of extern within the program or file:-**

**File name ex=> (program.c)**

```
#include<stdio.h>
int main()
{
extern int a; //calling that global variable
printf("%d",a);
return 0;
}
int a=12;. //Global variable
```

#### **Output:**

12

### **Use of extern keyword in another file:-**

**File name example=> program.c**

```
int a=12;
```

**File name example=> program2.c**

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

```
int main()  
{  
extern int a;  
printf("%d",a);  
return 0;  
}
```

**Output:**

12

\*\*\*\*\*



# 16.

## Command Line Arguments

**Q. What is Command Line Argument?**

**Ans=> *passing arguments to the program from the command line is called command line argument.***

- We all know that, values value which is passing to the function is called arguments.
- Here command line is CUI (Character/Command Used Interface) which is used by any professional programmer. For example Dos operating system is a command line because from here we can access computer by passing commands.
- If we are beginner in C Programming then we should go for IDE such as turbo c, vscode, etc. But professional programmer used command line from where they pass value to program by main() function.

**Q. Who call main() function?**

**Ans=>** we can call any used defined function directly or indirectly from main() function but here one more question arises that who call main function. The answer is operating system means main() function is called from operating system of computer. It is possible to pass arguments to the main

function of a program when we are calling it from command line.

**Q. What are the different ways of run any program?**

**Ans=>** there are three ways of run program:-

- 1) Using IDE=>** we can create program using **integrated development environment** like vscode, turbo c, etc.  
Each of the IDE provides **run** option to see the output as well as inputing values for program.
- 2) By double click=>** after generating .exe file we can also run program by clicking on that exe file. We can hold screen in that case by using getch() function at the end of the program.
- 3) Command line=>** this is the professional way of of running any program. We can handle our program from outside medium using Command line. In this concept we are passing instruction (value) to the program from operating system.

**\*\*Components of Command Line Arguments\*\***

There are 2 components (values) of Command Line Argument which are passing to the main function of the program:-

Syntax of main when we are passing values from Command line:- `main(int argc, char* argv[]);`

- 1) **argc**: argc stands for "arguments count". used to count number of arguments including **file name**. It counts number of arguments that's why it is of **int type**.
- 2) **argv**: argv stands for "arguments vector". It is used to receive input from Command line in the form of string. **argv** collects **n number of string** that's why it is **char pointer array type**.

**Program to initialize a structure by reading elements from Command line:-**

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

struct emp{
char* name;
long int MoNo;
float salary;
};

int main(int argc, char* argv[]) { //this is compulsory syntax
```

```
struct emp e1;

e1.name= argv[1];
e1.MoNo= atoi(argv[2]);
e1.salary= atof(argv[3]);

printf("Name of e1: %s", e1.name);
printf("Mobile Number of e1: %li", e1.MoNo);
printf("Salary of e1: %f", e1.salary);

return 0;
}
```

### \*\*\*Use of atoi() and atof()\*\*\*

We all know that arguments passing from the command line to main function is of **string type**. To convert these strings **int** and **float** type for performing some operations then we need to use **dos.h** header file functions which is:-

**1) atoi();** => stands for "automatic int". used to convert **string** into **integer type**.

Prototype: int atoi(char\* str);

Syntax: atoi(23);

**2) atof();** => stands for "automatic float". Used to convert string into float.

Prototype: int atof(char\* str);

Example: atof(23.45);

### Example for more clarity:

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

int main(int argc, char* argv[]); //this is compulsory syntax
if(argc>2) //indexing from 0
{
char* str1=argv[1];
char* str2=argv[2];
int x=atoi(s1);
int y=atoi(s2);

/* also written in this form */
/*
int x=atoi(argv[1]);
int y=atoi(argv[2]); */

printf("sum: %d\n", x+y);
}
else
printf("Insufficient input values");

return 0;
}
```

\*\*\*\*\*

## 17.

### Graphics in C

## Q. What is Graphics in C?

**Ans=>** Generally C programs runs only in a consol screen of any IDE that is a black Screen also called Character/ Command user interface (CUI). Using Functions of graphics.h header file we can develop GUI in place of CUI.

GUI screen is in Pixels form. And CUI is in Block form where each block holds one character.

## Q. How to initialize Graphic Mode in C?

**Ans=>** there are two types of modes available in C Programming one is CUI Mode and Second is GUI Mode.

There is no need to initialise **character user interface mode** because it is a default user interface in any programming language. But **graphic user interface mode** is need to be initialised and close.

to initialise and close **GUI Mode** there are two important functions of graphics dotage header file are used:-

**1) initgraph() :-** stands for "initialize graph". initgraph function or method is used to convert **CUI mode** into **GUI mode**. This function is used to to initialize **graphic mode or System**. This function is not return anything means void type.

**Syntax:** void initgraph(int \* graphdriver, int \*graphmode, char \*pathtodriver);

**Example:-** initgraph(&gd, &gm, "C:\\TC\\BGI");

Here initgraph is taking 3 arguments addresses which is need to be initialised before this function:-

i) **\*graphdriver:-** there are total 10 graphdriver available in C graphic. We can initialised graphdriver by numbers from 1 to 10 due to predifine Enumeration. If you don't know that which graphdriver is suitable for your system then you should go for **auto detect request** by assign **0** or **DETECT** to graphdriver.

ii) **\*graphmode:-** there are **n number of** graphics mode available in C graphic (with Enumeration) but if we are initialising graphdriver by **0** or **DETECT** then initgraph function automatically detect graphmode suitable for our program.

iii) **\*pathtodriver:-** path to drive means the path where given driver is located. This part is very essential to draw any diagram or to insert any colour on console.

- **c:\\tc\\bgi** contains n number of graphic related functions. Here **c** it is a folder in which **tc** named folder is available and after that in which **bgi** named folder is available in which all the functions and header files related graphics are stored.
- BGI stands for Boreland Graphics Interface which is a Bundle of graphics related libraries from where we can access all the pathdrivers.

## Q. How to initialize Graphic Mode in C?

2) **closegraph()** :- this function is used to close graphic mode which is previously initialised by initgraph() function.

After performing all the task in graphic mode or system it is mandatory to to close that graphics system. That's why we are using closegraph function.

**Syntax:-** void closegraph();

- This function is not taking any argument because it is automatically detect initgraph() function which is previously declared inside the program.
- This function is not return anything means void type

**Example for more clarity about initgraph & closegraph:-**

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

void main{
/* initializing graphdriver & graphmode */
int gd=0, gm;
initgraph(&gd, &gm, "c:\\tc\\bgi"); //initializing graphic mode
.....
.....
.....
.....
getch(); //for holding console for see output
closegraph(); //closing graph mode or system
}
```

**\*\*Some More essential Function in Graphic Mode\*\***



Accept `initgraph` & `closegraph` there are some more very important functions which are used to perform some activities in graphic mode or system.

I classified important functions into 4 types for easy to remember:-

- Shapes related functions in C graphic mode
- Colors related functions in C graphic mode
- Text related functions in C graphic mode
- Function for clear screen in C graphic mode

Now we are discussing about the functions of these four classifications:-

### **1) Shape related functions in C graphic mode:-**

There are n number of functions available in **graphics.h** through which we can draw different different types of shapes like circle, bar, rectangle, etc. Here we only discuss some important shapes functions.

**1. circle() :-** This function of `graphics.h` header file is used to draw circle.

**Syntax:** `void circle(int x, int y, int radius);`

**Example:** `circle(140, 45, 15);`

**2. bar() :-** this function of `graphics.h` header file is used to draw bar. Bar means the shape which is having 4 liner sides.

**Syntax:** `void bar(int left, int top, int right, int bottom);`

**Example:** `bar(50,20,80,100);`

**3. rectangle()** :- this function of graphic dot h header file is used to draw rectangle.

**Syntax:** void rectangle( int left, int top, int right, int bottom);

**example:** rectangle (200,10,310,75);

**4. line()** :- this function of graphics.h header file is used to draw line.

**Syntax:** void line(int x1, int y1, int x2, int y2);

**Example:** line(330,10,430,10);

**5. ellipse()** :- this function of graphics.h header file is used to create ellipse. ellipse is a long circle.

**Syntax:** void ellipse (int x, int y, int stangle, int endangle, int x radius, int y radius);

**Example:** ellipse(470, 42, 0, 360, 20, 35);

## **2) colours related functions in graphic mode:-**

Before discussing colours related functions first of all we are discussing about **colours in C graphics**.

There are total 16 colours available in C graphics especially in Dos based graphic. There are 0 to 15 predefined enumerated codes are available for accessing each of the colour easily. We can also use colour name in CAPITAL in place of predefined enumerated codes.

Following are the colours with code:-

BLACK	0
BLUE	1
GREEN	2
CYAN	3
RED	4
MAGENTA	5
BROWN	6
LIGHTGRAY	7
DARKGRAY	8
LIGHTBLUE	9
LIGHTGREEN	10
LIGHTCYAN	11
LIGHTRED	12
LIGHTMAGENTA	13
YELLOW	14
WHITE	15
BLINK	128

For using all the 16 (0-15) colours, graphics.h header file contains n number of functions. some of them are as follows:-

1. **setcolor():** This Function of graphics.h header file is used to set Color of each shape and text on console screen.

**Syntax:** void setcolor(int color);

**Example:** setcolor(2); or setcolor(GREEN);

2. **setbkcolor():** This Function of graphics.h header file is used to set background color.

**Syntax:** void setbkcolor(int color);

**Example:** setbkcolor(2); or setcolor (GREEN);

### 3) Text Related Functions in Graphic Mode:-

1. **Outtextxy() :-** This function of graphics.h header file is used to print any text message in Graphic mode. It print text in pixel format that's why it is taking x and y coordinate along with String.

**Syntax:** void outtextxy(int x, int y, char\* string);

**Example:** outtextxy(100, 200, "Dheeraj Patidar");

2. **settextstyle() :-**

this function of graphics.h header file is used to customise text font, text direction, and text size.

**Syntax:** void settextstyle( int font, int direction, int size);

**Example:** settextstyle( 1, 0, 3);

settextstyle function is taking 3 integer arguments first is font of text (0-4), second is direction of text(0 & 1), and third is size of font (1 to 10).

Some important points:-

- There are total 5 text fonts available in c graphics.  
Enumerated numbers of these five text fonts are from 0 to 4. 0 is used for default font.

DEFAULT_FONT	0
TRIPLEX_FONT	1
SMALL_FONT	2
SANS_SERIF_FONT	3
GOthic_FONT	4

- There are total 2 text directions available in c graphics.  
Enumerated number of these two directions are 0 and 1.  
Here 0 is used for default direction which is **left to right**.

Name	Value	Direction
HORIZ_DIR	0	Left to right
VERT_DIR	1	Bottom to top

- There are total 10 text size available in c graphics.  
Enumerated number of all text size is from 1 to 10.

Enumerated value	Size of boxes
1	8×8 size per box
2	16×16 size per box
3	24×24 size per box
4	32×32 size per box
5	40×40 size per box
6	48×47 size per box
7	56×56 size per box
8	64×64 size per box
9	72×72 size per box
10	80×80 size per box

**\*\*\* HAPPY ENDING \*\*\***