

INDEX

[INTRODUCTION](#)

[C PROGRAMMING](#)

[DATA TYPES](#)

[INPUT OUTPUT](#)

[OPERATORS](#)

[DECISION MAKING](#)

[LOOPS](#)

[ARRYS](#)

[CHARACTER ARRAYS](#)

[FUNCTIONS](#)

[STRUCTURES AND UNIONS](#)

[POINTERS](#)

[DYNAMIC MEMORY ALLOCATION](#)

[PREPROCESSOR DIRECTIVES](#)



PROGRAMMING

INTRODUCTION TO COMPUTER PROGRAMMING

What is Programming language :-

Language :- language is a tool to communicate with each other.

Program :- program is set of instruction.

Programming :- Method to apply programs to machine in order to do task.

Programming Language :- Language is used to communicate with machine for performing task in given order.

Computer only understand **Binary language** but for us speaking binary language is difficult, that's why programming languages invented to communicate with computer in human readable code and give instruction to perform task in given order.

Binary Language :- language that contain only two character 0 and 1 is binary language.

In binary language there are only two possibilities 0 or 1, true or false.

These binary language also called **Machine Language** or **Low level language**.

What is Low Level Language :-

The language that directly interact with **C.P.U.** or **Processor** is low level language.

- 1) **Binary Language.**
- 2) **Assembly Language.**

Assembly Language :- To make machine language easy to use, improvement over it using mnemonics.

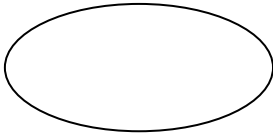

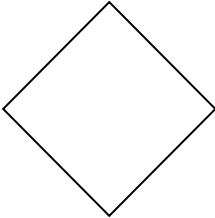


Assembler is used to translate mnemonics in machine code.

Example mnemonics :- Add, SUB, div, MUL, push, mov, and, call, printf, leave.

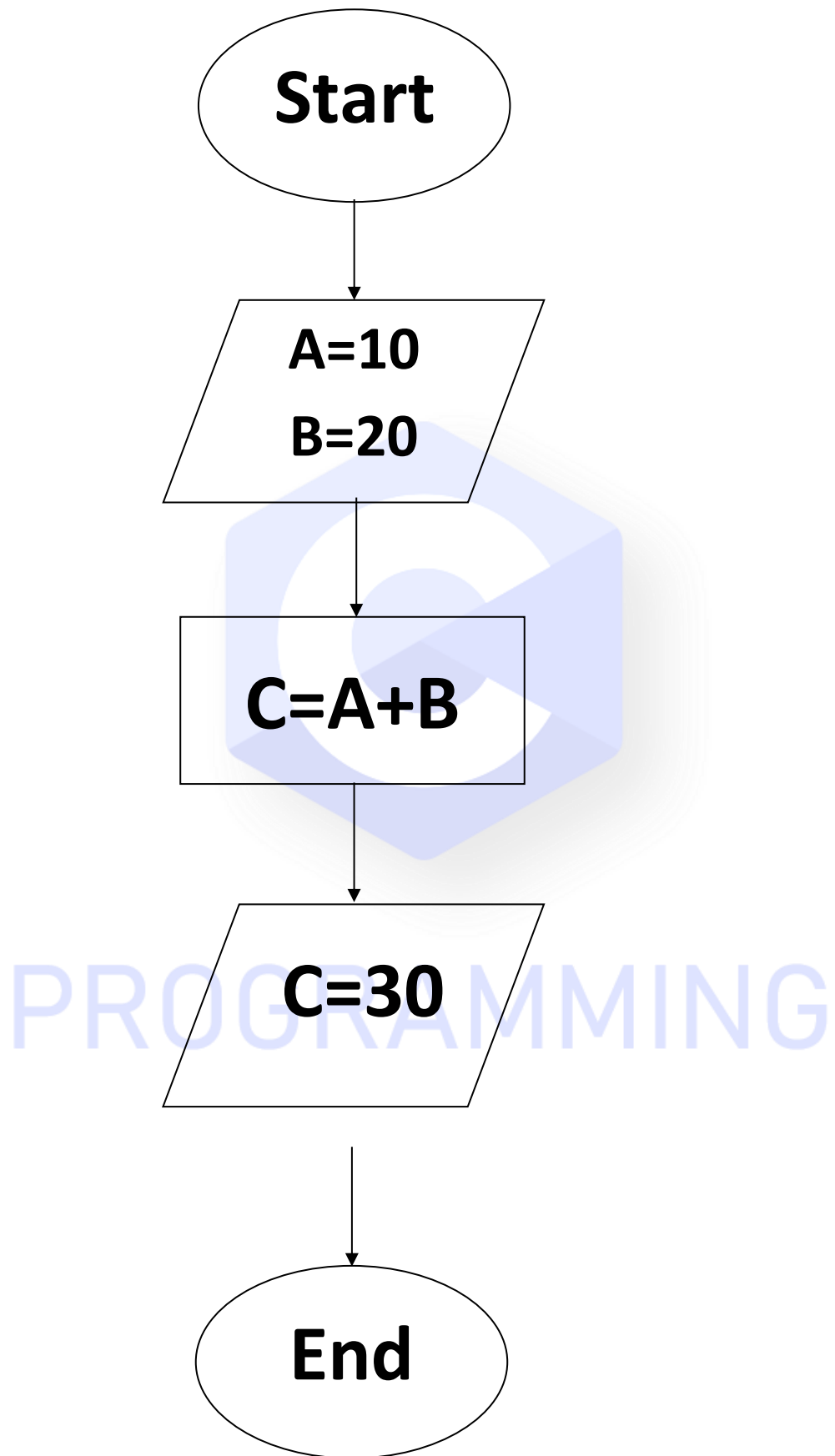
Assembly language is made coding more faster and understandable compare to give instruction in machine language. **Nasa's apollo 11 (july 16-24, 1969) moon lander also programed in assembly language.** Assembly language use to program drivers in now days.

FLOWCHART: -

Flowchart is the basic tool to understand programming logic. It is a graphical representation of a process. Each step of the process is indicated by a specific symbol. These symbols are connected by arrows indicating the direction of the process flow.

Flowchart	Symbol	Purpose
	Start/ End	The oval symbol represents the starting and ending of a program.
	Action/ Process	The rectangle symbol represents a single step or sub-process within a large process.
	Decision	The diamond box represents the decision. It is used to check the conditions that are used in the program.
	Input/ Output	This parallelogram is a symbol used to represent the input and output values.
	Arrows	The arrow represents a connection between the shapes and is used to define relationship.

Flowchart of adding two number:-



THE C PROGRAMMING LANGUAGE

EXPLORING C PROGRAMMING

Why to learn C language :-

The main objective of learning C language is logic building in programming also learning c helps to learn c++, when you learn C then half of more C++ syllabus is already covered. C language also uses there latency need minimal.

Latency :- The ability of computer system or network to provide responses with minimal delay. It measures in seconds & milliseconds.

It means when you press any key on keyboard for example you **PRESS C KEY** on keyboard and then it **DISPLAY OUTPUT** on your **Display** the time between pressing key to display output is latency. It is also used in network or internet system. It is called as ping then you click on any link on internet to time between you get output is latency.

Features of C language :-

1.C is known as middle – level language it combines controlling features of high level language like pascal and basic & flexibility features from low-level language.

2.The features of the low level language permits C to manipulate the memory in terms of bits, bytes and address.

3.C has numerous built-in data types, keywords, operators, functions & libraries that make it one of the most robust(Strong) programming Language.

4.**Portability :-** C is portable language that can write program on one system and run it to another system without making changes or minimal changes.

5.**Extendibility :-** In C language programmer can extend & add new functions & libraries to existing language. Also there are various built in libraries & however developers can add new functions according to their requirement.

USES OF C LANGUAGE :-

1. Linux operating system's most of the part built in C language.(unix)
2. MySQL database's most of part built in C language.
3. Cython C in python is built in C language, also Ruby ,Perl and python scripting languages are built in C language.
4. First compiler of **JAVA** is also built in C.
5. Popular OS like mac,windows,linux's kernel versions also built in C language.

Kernel :- it is system software which is part of operating system.

6. Libraries of python numpy,scipy are write in C language.
7. C language also use to supports other programming languages.
8. C is used in video game industries where speed and performance needs because of it's latency along with C++. C use most in developing game engines.
9. C & C++ programming is also use for solve competitive programming problem because there is time consistency in competition, your program can run fast when you use C & C++ in programming language.
10. Webserver called apache and nginx's most of part built in C language because of C language because of speed of C language.

HISTORY OF C LANGUAGE :-

First computer programming language was ALGOL (Algorithmic language) basic language of all **modern** programming language, after ALGOL language BCPL (Basic Combined Programming Language) is developed. It becomes popular computer language developed by Martin Richard in 1967 & update by Ken Thompson (designed and implanted original unix operating system 1971). Ken Thompson created another language based on BCPL is B language. B language used to create early version of unix operating system B had drawback that it didn't have data types.

Dennis M. Ritchie studied B language and decided to create new language that had all feature of B language. Therefore he named language C. It was developed in year 1972. At AT&T (American Telephone & Telegraph company) Bell Labs, In 1978 Brian Kernighan & Dennis Ritchie wrote book "The C programming language 1st edition", After publication of book C language known as "K&R C". This book remains standard book of C language.



In 1983 American National Standard Institute (ANSI) defined a new standard definition of C language, developed their standard of C language in 1988 made some changes in original design called it "ANSI C". In 1990, International Standard Organization (ISO) adopted the ANSI C which was also called the "ANSI/ISO C". In 1999, the standardization committee developed C99. However ANSI C continues to be most popular among developers & programmers. The development of C became the foundation for the development of object-oriented "C++ Language".

Year	Language	Founder and developer
1960	ALGOL	International Committee
1963	BCPL	Cambridge University
1970	B	Ken Thompson
1972	C	Dennis Ritchie

APPLICATION OF C LANGUAGE: -

Earlier, the main application of C language was for system development and creation of operating systems.

- Operating System
- Assemblers
- Language Compilers & Interpreters

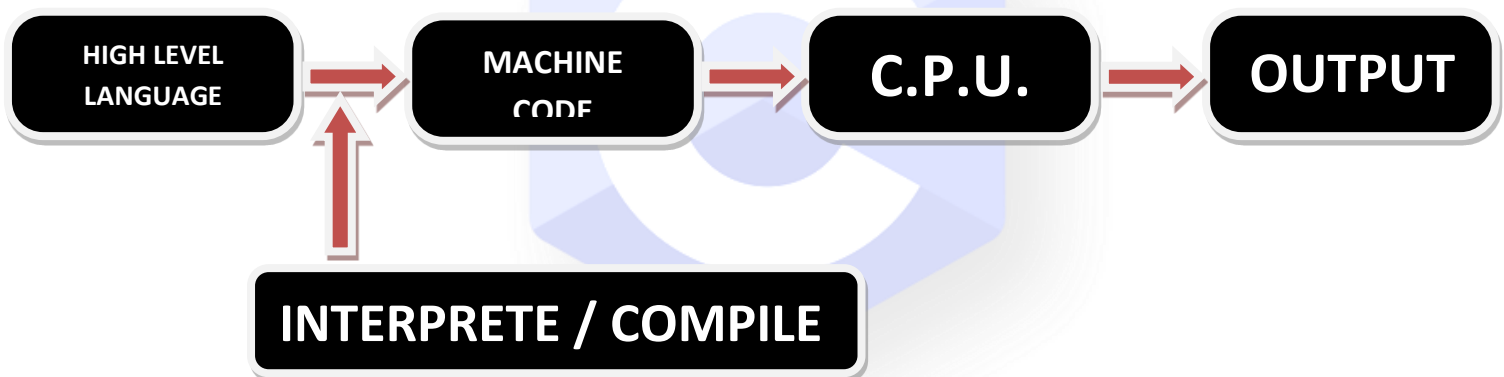
- Text Editors
- Network Drivers
- Data bases

COMPILATION AND EXECUTION OF C PROGRAM: -

Text editor: - Use to type and writing programs, program written in editors is called source program.

Compiler: - A programs are written in human readable form is called source code. To convert these sources codes in machine language for understand to computer. The compiler performs this task. Compilers covert High-level language to machine or assembly language.

Process of converting high level language to low level language :-



The high level language convert or translated to machine code then machine code interact to C.P.U. after completing processing by processor we get output.

Process of convert or translating high level language program into machine code is called compiling or interpreting

Compiling and interpreting are two different process of translating program to machine code the major difference between both is **Compiler is convert whole program at same time & Interpreter convert program statement by statement.**

There are two types of programming language low level and high level language.

LOW LEVEL LANGUAGE :-

- 1) Binary Language.
- 2) Assembly Language.

HIGH LEVEL LANGUAGE :-

C, C++ , c#, JAVA, PYTHON, JAVASCRIPT, etc...

COMPILER VS. INTERPRETER: -

Sr	INTERPRETER	COMPILER
1	Convert high-level language to machine code statement by statement or line by line.	Convert high-level language with whole source code to machine language.
2	Interpreter's execution speed is slow compare to compiler because it execute program statement by statement.	Compiler's execution speed is fast compare to interpreter because it execute whole program at time.
3	Interpreter execute program statement by statement that's why when error occur it generate error message immediately and stop remaining execution.	Compiler execute whole program at time that's why compiler generate error message after scanning whole program.
4	Interpreter display partial programming execution that before occurring errors.	Compiler did not display executed program until all errors sort out.
5	Showing error as it occur that's why debugging is fast and easy to compare compiler.	Debugging process is slow and hard finding error compare to interpreter.
6	Interpreter is always need everytime you run program.	Once program compile then it generate .obj or .o file after compiling program compiler does not require to run program.
7	languages used interpreter to execution of program is called scripting language .	Languages used compiler to execution of program is called programming language
8	To run program source codes always need that's why secrecy of codes can be compromised.	There is no need to source code to run program once's program compile, secrecy of codes is remain.
9	Examples :- Python, Javascript	Example :- C,C++

INSTALLATION OF TURBO C++: -

Turbo C is compiler that provides an Integrated Development Environment. Borland introduced it in 1987 and after some time replaced it with Turbo C++, which is use for write and execute C or C++ programs. Turbo C++ is available for free on the internet. Download Turbo C++ using following link.

<https://turboc.codeplex.com>

Installation of C compiler is very easy.

1. Download it from given link.
2. Extract the downloaded zip file.
3. Run "setup.exe" file.
4. Then after showing window of "Welcome", click on "Next >".
5. In License Agreement window click on "I accept the terms.....".
6. Then click on "Next >".
7. After that it will show default installation path and folder
8. Then click on "Install".
9. After completing Installation it will show Completed message.

DEVELOPING C PROGRAMS: -

Developing C programs in compiler requires the following steps: -

1. Writing programs.
2. Compiling programs.
3. Linking programs.
4. Executing programs.

Writing programs: - First step of developing programming is writing programming in human readable form with words and symbols. Programmers write and edit programs in text editors. Programmer saves C programs using .c extension this programs called source programs.

Compiling programs: - After writing C programs, these programs need to compile it using compiler, compiler first translates and then converts source programs in object codes. This object code stored in file with extension .o or .obj in subdirectory.

Linking programs: - C language has many different library routines whose linking is necessary if it is used to the programs. Linker performs the task of linking these library routines that generate an executable file with .exe extension.

Executing programs: - The last step of developing C program is run executable file and generates an output. The standard console of the operating system display output.

COMPILING: -



PROGRAMMING

STRUCTURE OF A C PROGRAM: -

Documentation Section	[Optional section]
Link Section	[Optional section]
Definition Section	[Optional section]
Global Declaration section	[Optional section]
User-defined Section	[Optional section]
main() function Section	[Compulsory section]

Documentation Section: - This section is for putting comments in the programs. It contains the name, author and a brief introduction of the program.

Link Section: - The link section contains the built-in library function of C language. The linker sends the instructions to the compiler to link the library functions from the system libraries.

Definition Section: - The definition section contains some pre-processor directives or symbolic constants.

Global Declaration Section: - In a program, A global variable that is declared outside all functions. If we declare int A, int B in a program in the global section, then they can be accessed throughout the program in C program by any function.

User-defined function Section: - The programmer needs some other functions along with main() function for writing programs. The programmer can put these functions in this section.

main() function Section: - The main function is most important and compulsory section of a C program. The function contains declarative and executable elements such as variables, data types, expressions, user-defined functions, etc. The execution of program start from this section and compiler executes all executable statements of the program.

A SAMPLE C PROGRAM: -

Let's run our first program in C++ turbo, software that we will use for C programming, but first we will make folder in turbo C++ for save our all programs in it.

1. First open turbo C++ software.
2. Open file menu with Alt + F.
3. Then go to Dos Shell by pressing D on keyboard.

4. It will open command prompt of turbo C++.
5. Type **md** or **mkdir** after that give space and type your file name. **e.g.** mkdir filename
6. After typing file name press Enter key.
7. To exit command prompt type 'exit' command.
8. Then open file menu.
9. And press 'N' to open new file.
10. And type following program.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    clrscr();
    printf("Hello World");
    getch();
}
```

11. Then press F2 to save program.
12. For save files enter in your folder.
13. Then rename file and give proper name to your program with '.C' extension.
14. Then press on 'OK'.
15. After saving program press 'Alt + F9' for compile program.
16. After compile if there is errors in program correct it, then compile it again.
17. Then press 'Ctrl + F9' to run program.
18. Then on console screen output will print.
19. For exit from console screen press any key on keyboard.
20. To see output press 'Alt + F5'.
21. Then press 'Alt + X' to exit turbo C++.

Let's see what we write in program to print 'Hello World'.

#include<stdio.h> #include<conio.h>	These are pre-processor commands.
#include	It is pre-processor directive to insert header files or libraries in programs. From that header file we will use functions.
<stdio.h>	It is one of header file. It stands for standard input and output file. .h is describe that it is header file.
<conio.h>	It is also another header file. Stands for console input and output.
main()	It is main function, C compiler execute main function from our program. Open and Close

	parenthesis (round brackets) shows it is function.
Void	Writing void before function name, it is describe return types of particular function. The return type of main function is void or none. It does not return anything to compiler.
{ }	Open and close curly brackets shows that whatever we write in those brackets it is in consider as main function's operation.
clrscr();	It is function for clear previous outputs from console screen. This function is include from conio.h file.
printf();	Printf is function to print output that we write in that function. This function is from stdio.h file.
"Hello World"	We write it in print function's parenthesis with double quotes as plain text.
getch();	It is another function from conio.h file get character input in program but people generally use it hold output on console screen.
;	It is one of important symbol in programming to terminate statement or end statement it like full stop after sentence.

COMMON GUIDELINES: -

- The developer should write every instruction in a separate statement.
- All statement should be formed in systematic order according to the logic of program.
- Every statement end with a semicolon [;] symbol.
- Small alphabet letters should be used for writing the statements.

CLASSIFYING DATA USING DATA TYPES IN C PROGRAMMING

C CHARACTER SET: -

Character set is building blocks for writing program in any programming language, these characters can be English alphabets, special symbols, or a digit for representing information

Character Set	Examples
Letters [Alphabets]	A,B,...Y,Z / a,b,...y,z.
Digits/numbers	0,1,2,3,4,5,6,7,8,9
Special characters	~!@#\$%^&*()_-={}[]:"'<>,?/.
White spaces	Blank space, Carriage return, Horizontal tab, New line.

Tokens: -

In programming language tokens are all the smallest individual units are the C tokens. C programming language has six type of tokens that are used to write program.

- Keywords
- Identifiers
- Constants
- Strings
- Special Symbols
- Operators

Token Name	Examples
Keywords	break, auto
Identifiers	name, price
Constant	50,-20,1
String	"ABC", "XYZ"
Special Symbol	[], @

Operators

+, *

KEYWORDS: -

Keywords are basic elements of programming, Keywords are reserved with special meaning. The C language supports 32 keywords, which written in lowercase.

ANSI keywords Supported in C

Auto	break	case	Char	const	Continue	default	Double
Do	enum	extern	Else	float	For	goto	if
Int	long	register	Return	short	Signed	sizeof	Static
Struct	switch	typedef	Union	unsigned	Void	volatile	while

IDENTIFIERS: -

Identifiers are user-defined names given to variables, functions, constant etc. in order to refer to them in program.

Rules for giving name to identifiers: -

- The first character of an identifier should start with an alphabet or underscore.
- The identifiers should contain alphabets, digits and underscore.
- There should not be any whitespace within the identifier.
- The keywords should not be used as identifiers.
- The first 31 characters are significant for an identifier.

Valid identifiers: -

- Add
- ODD
- _hello
- g1

Invalid identifiers: -

- 4hard
- char
- good@me
- hello C

COMMENTS: -

There are two types of comments: -

Syntax: -

- Single line comments: - `//` (double slash) e.g. `//This is comments`
- Multi line comments: - `/* */` (slash asterisk - asterisk slash) e.g. `/* this is comments`

Of multiple lines `*/`

Writing any text, numbers, and symbols in comments does not affect on programming execution. Comments are not case sensitive.

Use of Comments: -

- Comments are use for to write more information about programs
- Also use for write author name or date of programs etc.
- To hide unwanted programming code from execution or code that we will use later in our program.

DATA TYPES: -

Data types are use to define type of data we use in program it can be integers, floats, strings, characters.

There are two type of data types: -

- 1) **Built-in data types**
- 2) **User-defined data types**

Built-in data types: - Built-in data types are already defined in C language e.g. int, float, void, etc

User-defined data types: - data types that build with the help of built-in data types are user-defined data types e.g. array, pointers, structures etc.

INTEGER DATA TYPE: -

Integer data types represent whole numbers. It can be positive or negative numbers.

Data Type	Data Size [bytes]	Range
-----------	-------------------	-------

Int	2	-32,768 to 32,767 or 0 to 65,535
signed int	2	-32,768 to 32,767
Unsigned int	2	0 to 65,535
Short	2	-32,768 to 32,767 or 0 to 65,535
signed short	2	-32,768 to 32,767
unsigned short	2	0 to 65,535
Long	4	-2147483648 to +2147483648 or 0 to 4294967295
signed long	4	-2147483648 to +2147483648
unsigned long	4	0 to 4294967295

FLOAT DATA TYPES: -

Float data type represent real numbers. The float, double and long double is three float data type provided by C programming language.

Data Type	Data Size [Bytes]	Range
Float	4	-3.4e38 to +3.4e38
Double	8	-1.7e308 to 1.7e308
long double	10	-1.7e4932 to 1.7e4932

CHARACTER DATA TYPE: -

The Character data type represents characters and strings. Char is the character data type of C language.

Data Type	Data Size [Bytes]	Range
Char	1	-128 to 127 or 0 to 255
signed char	1	-128 to 127
unsigned char	1	0 to 255

VOID DATA TYPE: -

Sometimes, program needs a data type for representing no values, in C programming language, the data type void not return any value. Mainly use for functions and pointer did not return value.

Command to check data type: - `printf("%u",sizeof(int));`

FORMAT SPECIFIERS: -

Format specifiers describe which type of data we use for printing standard output or taking standard input. Symbol "%" is used for declare specifiers.

Data Type	Format Specifiers
Int	%d
Signed int	%d or %i
Unsigned int	%u
Short	%d
Signed short	%d
Unsigned short	%u
Long	%ld
Long long	%lld
Float	%f
Double	%lf
Long double	%Lf
Float or double exponential format	%e or %E
Char	%c
Octal value	%o
String[Array of characters]	%s
Hexa value	%x or %X

VARIABLES: -

A variable is a named place holder defined to store data temporarily. It is named location in memory that holds value and can be modified by the programs.

Variables stored inside functions are local variable, outside the functions are global and in definition of functions parameters are formal parameters.

NAMING OF VARIABLES: -

- A variable name should contain alphabets, digits and underscore.
- The name of a variable should start with an alphabet or underscore.
- There should not be any whitespace within the variable name.
- The keywords should not be used for as variable names.
- C programming language is case-sensitive, hence uppercase and lowercase characters consider differently. E.g. "int a" & "int A" this two variables are different in C.

Valid names: -

- Goods
- _head
- Had_mad
- Glory11

Invalid names: -

- Goto [It is keyword]
- 4heaven
- Hello ride
- God%md

DECLARATION OF VARIABLES: -

Syntax: -

- Declaring variable type 1 :- data_type variable_name;
- Declaring variable type 2 :- data_type var_name1, var_name2, var_name3;

Variable Declaration	Description
<code>int a;</code>	Declares an integer variable for storing numbers
<code>int p, q, s;</code>	Declares integer variables for storing numbers
<code>float f;</code>	Declares a float variable for storing number real numbers

double avg_number;	Declares a double variable for storing real number, which is long in precision
short maths_number;	Declares a short integer variable for storing a number
char ch;	Declares a character variable for storing a character
long salary;	Declares a long integer variable for storing salaries

ASSIGNING VALUES TO THE VARIABLES: -

Assignment operator (=) is use for assigning value to variable.

Syntax: -

- After declaration :- datatype var_name; // Declaration
 var_name = value; // Assignment
- At the time declaration :- datatype var_name = value; // both are at same time

Variable Assignment	Description
int a; a = 10;	Declaring and assigning the value [10] to the integer variable 'a'.
Int p = 20;	Declaring and assigning an integer value to a variable
Float f = 24.8;	Declaring and assigning the float variable 'f'
double pi; pi = 3.14;	Declaring a double variable and assigning a value to it.
Char ch = 'A';	Declaring and assigning a value 'A' to the character variable 'ch'.

CONSTANTS: -

Constant is value that does not change during execution.

Types of Constants

Integer constant: - It store value that does not have decimal point. C language allows use suffixes such as U or L for show unsigned or long constants.

Examples: -

- 110
- +89L
- -6723
- 2318U

Floating constant: - It stores real value or value with decimal point. The suffix F or f represent floating constant.

Examples: -

- 271.33F
- +65.45
- -47.2389
- 5006.34L

Use integer part and fractional part for representing the exponential form both part separated by a letter 'e'. value **before 'e'** are called **mantissa** and value **after 'e'** are called **exponent**

Examples: -

- 5.2e4
- +7.6e78f
- -0.32e+8
- -0.9e-6

Character/String constant: -

It stores characters or strings. Character constant should written in single quote [' '] and string constant should written in double quote [" "].

Examples: -

- 'd'
- "sample"

NAMING OF CONSTANT: -

- The name of the constant should start either with an alphabet or with an underscore.
- The constant name should only contain alphabets, digits and underscore.
- There should not be any whitespace within the constant name.
- The keywords cannot be used as constant names.

Examples: -

- Age
- Bank_Name

DECLARING AND ASSIGNING VALUES TO CONSTANT: -

Declaring and assigning values to constant.

Syntax: -

- Assign and declare value:- `Const datatype constant_name = value;`
- Assign value in pre-processor:- `#define identifier data = value;`

Constant	Description
<code>Const int h =10;</code>	Declares and assign a value 10 to the integer constant 'h'
<code>Const float pi = 3.14;</code>	Declares and assigns a value 3.14 to the float constant 'pi'
<code>Const Char j = 'A';</code>	Declares and assigns a constant value
<code>#define pi 3.14</code>	Declares a constant using the #define pre-processor

ENUMERATED DATA TYPE: -

It is user defined data-types that enable programmer to create custom data types. It define list of keywords that makes program more understandable. It contain integral constant.

Syntax: -

- `enum enum_name { name1=1, name2=5,name3=7};`
- `enum enum_name { value1, value2, value3 }; // It takes default value for 0,1,2, etc.`

STORAGE CLASSES: -

Every C variable has storage class, there are four type of storage class in c: -

- auto
- register
- static
- extern

The auto storage classes: -

It is the default storage class for all local variables within c program.

Example:-

```
{  
int days;  
auto int week;  
}
```

The register storage class: -

This class defines those variables that can be stored in register instead of RAM, maximum size of variable is equals to register size which is usually one word and it can't have the unary '&' operator as it does not have memory location. It define using 'register' keyword.

Example: -

```
{  
Register int length;  
}
```

Register variable might get stored in register if it has any memory constraint depending on the hardware.

The static storage class: -

This storage class instructs compiler for keeping local variable in existence until program exist instead of creating and destroying it every time it is declared and it comes and goes into the scope.

The static modifiers can also be applied on global variable.

The extern storage class: -

This class used for giving reference of a global variable to all the program files. Extern points to the variable name at a storage location which has been previously defined.

Extern is always declare a global variable or function in another file. It is commonly used when two or more files share the same global variables or functions.

MANAGING INPUT AND OUTPUT OPERATIONS

HEADER FILES: -

A header file has the extension **.h** it contains various function definitions. There are two types of header files available first is **user / used defined header files** programmer write and second **system / standard library header files** which come with compiler.

MACROS: - It is name given to block of statements used as a pre-processor directive, its values did not change during program. It is defined with pre-processor directive **#define**.

Example: - `#define pi 3.14, #define AREA (a) (a*a) ----- int a1=5 , area; area = AREA(a1)`

GLOBAL VARIABLES: - These variables are defined outside a function and can be accessed throughout the program by any function.

FUNCTION PROTOTYPES: - It includes function signature, function name, return type.

STANDARD C LIBRARY FUNCTIONS: - These are inbuilt functions in C libraries their function prototype and data declaration present in respective libraries, when these functions need in program, their header files should include for access.

Header	Standard Library Functions
assert.h	Diagnostic functions
ctype.h	Character handling functions
locale.h	Localization functions
math.h	Mathematics functions
setjmp.h	Nonlocal jump functions
signal.h	Signal handling functions
stdio.h	Input/Output functions
stdlib.h	General utility functions
string.h	String functions

Syntax: -

- `#include <file>` `//this statement include system header file in program.`
- `#include "file"` `//this statement include user defined header file in program.`

INTRODUCTION TO THE CONSOLE INPUT/OUTPUT FUNCTIONS: -

All function that receive input from the keyboard and write output to the output device [VDU(Video Display Unit)/Screen] are called Console I/O functions. The Keyboard and screen together are known as the console.

Formatted and unformatted are the two major categories of the console input/output functions.

Formatted functions: - are the functions that perform input and output operations with required formatting.

Unformatted functions: - are the functions that do not perform any formatting in the input and output.

Categories of input/output functions: -

- Formatted input function – `scanf()`
- Formatted output function – `printf()`
- Unformatted input function – `getchar()`, `gets()`, `getch()`, `getche()`,
- Unformatted output function – `putch()`, `putchar()`, `puts()`

Formatted function did show details as 'where this output appear on screen, number of places after decimal numbers.

FORMATTED INPUT FUNCTION: - `scanf()`;

This function defines any input data, which arranged in specific format. This function use for provide input in a fixed format. It work with all data types such as int, float, double or char means using this function can input numeric, real, character data.

Syntax: -

`scanf("format string", the list of addresses of variables)`

Different Forms of the <code>scanf()</code> function	Exploration
<code>Scanf("%d", &n)</code>	The function will take a number from the keyboard, as the format specifier is %d. The function will allocate an address [2 bytes] for the variable "n" using &n in the memory.

Scanf("%2d", &n)	The function will take a number from the keyboard, as the format specifier is %d. The function will allocate an address [2 bytes] for the variable "n" using &n in the memory. If the value 348 is entered then it will take only 34 as the width specified is up to 2 positions.
Scanf("%f", &m)	The function will take a real number from the keyboard, as the format specifier is %f. The function will allocate an address [4 bytes] for the variable "m" using &m in the memory.
Scanf("%c", &n)	The function will take a character from the keyboard, as the format specifiers is %c. The function will allocate an address [1 byte] for the variable "n" using &n in the memory.
Scanf("%d %d", &a, &b)	The function will input two numbers from the keyboard, as the format specifier is %d two times. The function will allocate two address [2 bytes for each] for the variable "a" and "b" using &a and &b respectively in the memory.
Scanf("%d %f", &x, &y)	The function will input two numbers where one number is an integer and another is a real number from keyboard, as the format specifier is %d and %f. The function will allocate two addresses [2 bytes for the integer and \$ bytes for the real number] for the variable "x" and "y" using &x and &y, respectively in the memory.
Scanf("%d %f %c", &n, &m, &p)	The function will input three inputs including integer, float and character from the keyboard, as the format specifier is %d, %f, and %c. The function will allocate three addresses [2 bytes for an integer, 4 bytes for real number and 1 byte for the character] for the variable "n", "m" and "p" using &n, &m, and &p, respectively in the memory.

FORMATTED OUTPUT FUNCTION: - printf();

It is most common and simple output function. It can print output according to types of data. It can also print simple text labels.

Syntax: -

printf("Format string", the list of variables)

ESCAPE SEQUENCE: -

In printf function escape sequences are used for take output in new line, blank tabs, etc. it is optional to use escape sequences. It uses backslash with character or symbols.

Escape Sequence or Character	Name	Use
<code>\n</code>	New line	For moving to the new line
<code>\t</code>	Tab	For creating space between variables
<code>\b</code>	Backspace	For moving the cursor one position to the left from its current position
<code>\r</code>	Carriage return	For moving cursor to the beginning of the line from the current position
<code>\a</code>	Alert	For alerting the user through sounding the speaker within the computer
<code>\f</code>	Form feed	Moving the computer stationary attached to the printer to the top of the next page.

Examples: -

Different Forms of the printf() Function	Explanation
<code>printf("Formatted input output functions")</code>	The function prints the text "Formatted input output functions" to the console [screen].
<code>printf("Formatted input output functions \n")</code>	The function prints the text "Formatted input output functions" to the console [screen] and the cursor will move to the next line.
<code>printf("Ram score %d in Maths",num)</code>	The function prints the output as, "Ram score 89 in Maths" if the user enters the value 89 in the variable "num".
<code>printf("Any real number = %f", avg)</code>	The function prints the output as, "Any real number = 78.34" if the user enters the value 78.34 in the variable "avg".
<code>print("A single alphabet = %c", ch)</code>	The function prints the output as, "A single alphabet = a" if the user enters the value 'a' in the variable "ch".

printf("A string = %s", s1)	The function prints the output as, "A string = sample" if the user enters the value "sample" in the variable "s1".
Print("M = %d \t N = %f", m, n)	The function prints the output as, "M = 10 N = 20.33" if the user enter the values 10 and 20.33 to the variables m and n respectively.

UNFORMATTED INPUT FUNCTIONS: -

This functions work with a single character and string of characters. getchar(), getch(), getche(), and gets() are unformatted functions. These functions read a character or set of characters from standard input device such as keyboards.

Getchar() Function – Reading a Single Character: -

This function inputs or reads a single character from the standard input device, keyboard.

Syntax: -

```
variable_name = getchar();
```

Example: -

```
char ch;
ch = getchar();
printf(ch);
```

getch() and getche() Functions: -

The function getchar() hold program until inputting character and pressing "Enter Key".

These both functions are alternative to getchar() function available in header file "conio.h". The function getch() inputs a character from the keyboard but keyed will not enclose on screen. That means when you press key as input in getch() function it will not display in input. And in getche() function will show entered input display on screen. And these functions read single character and take input without press "Enter key". The getch() echoes the input character on the screen.

Syntax: -

```
variable_name = getch();
```

```
variable_name = getche();
```

Example: -

```
char ch;  
ch =getch();
```

gets()Function – Reading a String: -

The gets() function input and reads the string from standard input devices.

Syntax: -

```
gets(str)
```

Example: -

```
char name [size_of_str];  
gets(name);
```

UNFORMATTED OUTPUT FUNCTIONS: -

There is three unformatted output functions are putchar(), putchar(), and puts(). These functions write output character or set of character on screen.

putchar() Function – Writing a Character

It is similar function to getchar() function but writes the output on screen. This function available in header files “stdio.h”. The putchar() function also similar to putchar() that writes a character to screen.

Syntax: -

```
Putchar(character)
```

Example: -

```
char t='R';  
putchar(t);
```

puts()Function – Writing a String: -

This function also similar to gets() but it display string output on screen. it does not need formatting like printf() function it print string as it is.

Syntax: -

```
puts(str)
```

Example: -

```
char name[size_of_str]="Computer";  
puts(name);
```

PERFORMING MATHEMATICAL AND LOGICAL OPERATIONS: OPERATORS AND EXPRESSIONS

OPERANDS AND OPERATORS: -

Operator: - it is symbol that takes some values and performs some calculation and does some operations on those values.

Operand: - The values on which an operator perform calculation and does some operations are called **operand**.

There is some built-in operators for doing simple and complex mathematical and logical calculations. These operators work with data types including numbers, real numbers and characters. There is list of these operator groups.

- Arithmetic operators
- Relational operators
- Logical operators
- Bitwise operators
- Assignment operators
- Conditional operators
- Other operators

TYPES OF OPERATORS: -

Arithmetic Operators: - This is most common group of operators. These operators perform normal mathematical operations as addition, subtraction multiplication and division.

Operator	Operator Name	Syntax	Operation
----------	---------------	--------	-----------

+	Addition	Operand1 + Operand2	It adds two operands.
-	Subtraction	Operand1 – Operand2	It subtracts one operand from another operand.
*	Multiplication	Operand1* Operand2	It performs the multiplication on two operands.
/	Division	Operand1/ Operand2	It performs the division and truncates the fractional part after division.
%	Modulus	Operand1% Operand2	It performs modulus division and generates the remainder after the division operation.

Operator	Example
+	3 + 4 = 7
-	10 – 3 = 7
*	6 * 4 = 24
/	9 / 2 = 4 or 16 / 2 = 8
%	9 % 2 =1 or 16 % 2 = 0

Relational Operators: - These operators are used for comparing values.

Operator	Operator Name	Syntax	Operation
<	Less than	LeftOperand < RightOperand	It checks whether the left operand is less than the right operand or not, if yes, then it returns true [1] otherwise zero [0].
>	Greater than	LeftOperand > RightOperand	It checks whether the left operand is greater than the right operand or not. If yes, then it returns true [1] otherwise zero [0].
<=	Less than equal to	LeftOperand <= RightOperand	It checks whether the left operand is less than or equal to right operand or not. If yes, then it returns true [1] otherwise zero [0].
>=	Greater than equal to	LeftOperand >= RightOperand	It checks whether the left operand is greater than the right operand or not. If yes, then it returns true [1] otherwise zero [0].
==	Equal to	LeftOperand == RightOperand	It checks whether the left operand is equal to the right operand or not. If yes, then it returns true [1] otherwise zero [0].

!=	Not equal to	LeftOperand != RightOperand	It checks whether the left operand is equal or not to the right operand. If yes, then it returns true [1] otherwise zero [0].
-----------	--------------	-----------------------------------	---

Operator	Example
<	3 < 4 = 1; 10 < 1 = 0
>	3 > 4 = 0; 10 > 1 = 1
<=	3 <= 4 = 1; 10 <= 1 = 1
>=	3 >= 4 = 0; 10 >= 1 = 1
==	3 == 4 = 0; 10 == 10 = 1
!=	3 != 4 = 1; 10 != 10 = 0

Logical Operators: - are used in situations where a programmer needs to perform more than one comparison at the same time for getting output.

Truth table for “AND” and “OR” operators: -

Operands		AND	OR
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Truth table for “NOT” operator :- It is a negation operator which inverse given condition.

A	!A (NOT A)
0	1
1	0

In logical operator 0 indicates false and 1 indicates true.

Operator	Operator Name	Syntax	Operation
&&	Logical AND	Operand1 && Operand2	It returns true if both operands contain nonzero value or 1 otherwise it returns false.
 	Logical OR	Operand1 Operand2	It returns true if any one operand contain nonzero value or 1 otherwise it return false.

!	Logical NOT	!operand	If a condition is true, then Logical NOT operator will make it false and vice versa.
---	-------------	----------	--

Operator	Example
&&	3 && 4 = 1; 3 && 0 = 0
	3 4 = 1; 3 0 = 1
!	!3 = 0; !0 = 1

Bitwise Operators: - These operator work on bits. It first converts given numbers into bits and then performs operation.

Operator	Operator Name	Syntax	Operation
&	Bitwise AND	Operand1 & Operand2	It covert both operands in bits and compare with each other. If bits of both operands is one then it return 1 bit otherwise zero. And then it shows output.
	Bitwise OR	Operand1 operand2	It covert both operands in bits and compare with each other. If bits of one of operands is one then it return 1 bit otherwise zero. And then it shows output.
^	Bitwise exclusive OR	Operand1 ^ Operand2	It covert both operands in bits and compare with each other. If bits of both operands is different then it return 1 bit otherwise zero. And then it shows output.
~	Negation	~Operand	It performs the complement operation.
>>	Right Shift	Operand1 >> Operand2	It shifts the bits of the operand1 to the right by the number of bits defined in the operand2
<<	Left Shift	Operand1 << Operand2	It shifts the bits of the operand1 to the left by the number of bits defined in the operand2

Operator	Example
&	4 [100] & 3 [011] = 0 [000]
	4 3 = 7 [111]
^	4 ^ 3 = &[111]

~	~4 = -5
>>	4 >> 2 = 1
<<	4 << 2 = 16

Assignment Operators: - These operators are used for assigning values to the variables, constants and identifiers.

Operator	Operator Name	Syntax	Operation
=	Simple assignment	Operand1 = value	It assigns the values to the operand1
+=	Add AND assignment	Operand1 += Operand2	It adds both operands and assigns the value to the Operand1
-=	Subtract AND assignment	Operand1 -= Operand2	It subtracts both operands and assigns the value to the Operand1
*=	Multiply AND assignment	Operand1 *= Operand2	It multiplies both operands and assigns the value to the Operand1
/=	Division AND assignment	Operand1 /= Operand2	It performs division between both operands and assigns the value to the Operand1
%=	Module AND assignment	Operand1 %= Operand2	It performs module division between both operands and assign the value to Operand1
&=	Bitwise AND assignment	Operand1 &= Operand2	It performs bitwise AND operation between both operands and assigns the value to the Operand1
=	Bitwise OR assignment	Operand1 = Operand2	It perform bitwise OR operation between both operands and assigns the value to the Operand1
^=	Bitwise XOR assignment	Operand1 ^= Operand2	It performs bitwise exclusive OR operation between both operands and assigns the value to the Operand1
>>=	Right shift assignment	Operand1 >>= Operand2	It performs the right shift operation between both operands and assign the value to Operand1
<<=	Left shift assignment	Operand1 <<= Operand2	It performs the left shift operation between both operand and assign value to the

Operator	Example
+=	x+=y is same as x=x+y

-=	x-=y is same as x=x-y
=	X=y is same as x=x*y

Increment and Decrement Operators: -

Increment [++] and decrement [--] operators are very useful operators for building any application. These operators are unary operators that work with single operand. At the same time these operators either add or subtract the value by one.

Operator / Syntax	Operator Name	Operation
++Operand1	Pre-increment	It first increments the operand value and then assigns it into the operand.
--Operand1	Pre-decrement	It first decrements the operand value and then assigns it into the operand.
Operand1++	Post-increment	It first assigns the value to the operand and then increments the operand by 1.
Operand1--	Post-decrement	It first assigns the value to the operand and then decrements the operand1

Example	Explanation
++A + ++B Where A = 1 and B = 1	=> ++1 + ++1 => 2 + 2 = 4 Here it first increments A by 1 then returns 2. It also increments B by 1 then return 2. Now it adds 2 + 2 and return 4. At last, A=2, B=2.
X++ + Y++ Where X = 1 and Y = 1	=> 1++ + 1++ => 1 + 1 = 2 Here it takes the value of X and Y, then increments by 1. After addition, it returns 2. After post increment operation in X and Y, we get X = 2 and Y=2.

Conditional Operators: - [?:]

It is ternary operator that works with three operands. It provide for check more than one condition.

Syntax: -

Expression1?Expression2:Expression3.

Example: -

a>b (condition) ? a (if true) : b (otherwise).

Other Operators: -

Operator	Syntax	Operation
Comma [,]	Operand1, Operand2, ...	It separates the operands and the compiler performs or selects the operator from left to right.
sizeof()	sizeof(Operand)	It specifies the size of the operand. For example, if the operand is an integer then it returns 2 bytes.
Pointer[*]	*Operand	It represents the variables as pointer variable.
Address off [&]	&operand	It returns the address of the operand.

Operator	Example
,	a, b, c Here, the comma operator separates three variables. The compiler first evaluates the first variable a then second variable b and then variable c.
sizeof()	int a; Sizeof(a) = 2 Here, a is an integer variable and sizeof(a) would return the 2 bytes.
*	int *p; Here, *p denoting that p is an integer pointer variable.
&	Int *t, d; t = &d; Here, *t is an integer pointer variable and d is normal integer. The variable t is storing the address of variable d.

OPERATOR PRECEDENCE AND ASSOCIATIVITY: -

Operator precedence finds out the order in which operators are executed within the expression. There is a possibility that an expression may contain operators that have the same precedence. In this case compiler may get confused, which one to execute first. To resolve this there is operators associativity. It determines the order in which operators having the same precedence are executed. It is can either “left to right” or “right to left”.

Operators	Associativity
() []	Left to Right

+ - [Unary plus and minus] ++ -- ! ~ *[Pointer] &[Address of] sizeof() type cast	Right to left
* / % [Binary operators that perform arithmetic calculation]	Left to Right
+ - [Binary operators plus and minus] that perform arithmetic calculation	Left to Right
<< >>	Left to Right
< > <= >=	Left to Right
== !=	Left to Right
&	Left to Right
^	Left to Right
	Left to Right
&&	Left to Right
	Left to Right
?:	Right to Left
Assignment operators	Right to Left
,	Left to Right

EVALUATION OF EXPRESSIONS: -

An arithmetic expression is a combination of operators, operands and parenthesis. The C compiler evaluates the expression according to priority, precedence and associativity of the operators.

In an expression, operator always comes between two operands. Here are some examples of arithmetic expressions: -

1) $a+b*c/d$, 2) $a*a+2*b*c+c*c[a^2 + 2bc + c^2]$

Example: -

$X=10*2+50/5$ operator * will perform first as the associativity of binary operator is left to right.

$X=20 + 50/5$ Now in + and / operator / has highest precedence.

$X=20 + 10$

now remaining operator will perform.

$X=30$

CONTROLLING THE PROGRAM ORDER: DECISION MAKING

IF STATEMENT: -

Simple if statement is first decision-making statement. It takes one or more conditions. It checks condition if it is true it execute given statement otherwise it exit from if statement.

Syntax: -

```
If(condition)
{
    Statement;
}
```

Example: -

```
If(n > 0)
Positive number
```

IF...ELSE STATEMENT: -

It is second decision-making statement, the “if” part of this statement contain condition, if the condition is true then it will execute the statement from if part otherwise statement contain in else part.

Syntax: -

```
If(condition)
{
    True statement block
}
Else
{
    False statement block
}
```

Example: -

```
If (age > 18)
    Eligible
Else
    Not eligible
```

NESTED...IF STATEMENT: -

Sometimes needs to check certain conditions only if some previous condition is checked, in that case nested if statement used as condition within condition.

Syntax: - nested if statement

```
If (condition1)
{
    If(condition2)
    {
        Statements
    }
    Statements
}
```

Syntax: - nested if...else statement

```
if(condition1)
{
    if(condition2)
    {
```



```

        True Statements Blocks
    }
    Else
    {
        False statements blocks
    }
}
Else
{
    False statements blocks
}

```

ELSE...IF LADDER STATEMENT: -

This statement chain ladder of if else statement when you need to check multiple conditions and make multiple decision according to them then you will need else if statements at that time, the execution of else if start from top and goes to down, if all condition false then it execute default statement.

Syntax: -

```

if(condition1)
    Statement1;
else if(condition2)
    statement2;
.
.
.
else if(condition n)
    statement n;
else
    default statement;

```

SWITCH STATEMENT: -

This statement helps to design multi-way decision statement. The keyword “switch” is used to define statement. The condition of switch statement is known as case labels. It can be integers, characters or constants contain some values. All values should unique that ends with colon [:] if value given value match with case labels then it execute statement and exit from switch statement.

“Break” keyword is used with case labels to exit after case label matching.

Syntax: -

```

switch(condition)
{
case value1:
    statement
    break;
case value2:
    statement
    break;
...
...
default:
    default block
    break;
}

```

Points need to consider during using switch statement: -

- The case label or expression should have an integer, constants or character.
- Each case label should end with a colon.
- All case labels should have same datatype.
- “Break” keyword need to exit switch statement after case executed, without it all other case statements after matching case will executed by default

NESTING OF SWITCH STATEMENT: -

It is switch statement can check conditions within switch statement.

Syntax: -

```

switch(condition1)
{
    case value1:
        statement
        switch(condition2)
        {
            Case value1:
                statement
                break;
            case value2:
                statement
                break;
        }
        Break;
    Case value2:

```

Statement

...

THE ?: OPERATOR: -

This is decision-making statement. This is conditional operator that expression before symbol '?' is condition and if it true expression before symbol ':' is executed otherwise expression after symbol ':' executed.

Syntax: - "Expression1? Expression2: Expression 3"

Nesting conditional operators :-

Syntax: -

"expression1? "expression1? expression2: expression3":expression3"

Expression1 is condition, if it is true then compiler go to nested expression1 and check it otherwise it will go to expression3.

REPEATING SEQUENCE OF INSTRUCTIONS: LOOPS

BASIC CONCEPT OF LOOPING: -

In programming languages, looping means to execute a set of statement for a specified number of times or till a condition remain true. The compiler will check condition in loop if it is true it will executes statements inside loop again and again until condition is false.

Loop has three major parts; initialization, loop condition, and increment/decrement loop counters. Initialization is part that contain where will loop start, loop condition contain where will loop terminate or how many times it will executes. Increment/decrement part maintains the count to check that loop has executed the exact number of times. It is variable it can increment or decrement as required for program.

WHILE LOOP: -

It is simple loop that execute one or more statements until condition is true. The conditions of it placed in the parenthesis. The compiler first checks the looping conditions and then executes the loop statements; hence, it is also called entry-controlled loop. Means It checks condition before entering loop then looping begins.

Syntax: -

Loop initialization
while(condition)

```
{  
    Statement(s);  
}
```

Example: -

```
i = 1  
while(i<10)  
{  
    printf("while loop");  
    i=i+1;  
}
```

DO...WHILE LOOP: -

It is same as while loop but it have one difference than while loop, condition is checked at end of loop in do...while loop that's why this loop will execute at least one time apart from this everything is similar in do...while loops.

Syntax: -

```
loop initialization  
do  
{  
    Statement(s);  
}  
While(condition(s));
```

Example: -

```
i = 1  
do  
{  
    Printf("While loop");  
    i = i + 1;  
}  
While(i<10);
```

FOR LOOP: -

This loop contains initialization, looping condition, and loop counter. There is initialization after keyword "for", after it looping or termination condition and then increment or decrement counter and then statement executed until condition is true.

Syntax: -

```
For(initialization; looping condition; increment/decrement)
{
    Statement(s);
}
```

Example: -

```
For(i=0;i<10;i++)
{
    Printf("For loop");
}
```

Features: -

- All three essential parts of looping place in line that reduces lines of code.
- Initialization and counters can be two in single loop for example (i=0,j=0;i<10;i++,j++) also it is not necessary that use all initialization variable in looping condition.
- Comma [,] operator is used to separate more than one initialization variables and loop counter variables.
- You can omit to write initialization and loop counter parts like for(;i<10;) in this case initialize variable can be before loop and counter variable in body of loop.
- Also for loop can use for(;;) in this way. It will infinite loop if break statement did not used in body of loop.

NESTING OF LOOPS: -

The loop which defined within loop is called the inner loop, and loop contains another loop is called outer loop.

Syntax: - nesting while loop

```
Loop initialization
While(condition)
{
    Statement(s);
    While(condition)
    {
    }
}
```

Syntax: - nesting do...while loop

Loop initialization

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

syntax: - nesting for loop

```
for(initialization, condition, increment/decrement)
{
    statement (s);
    for(initialization, condition, increment/decrement)
    {
        Statement (s);
        ... ..
    }
    ... ..
}
```

LOOP CONTROL STATEMENTS: -

Sometime it became necessary to jump from middle of loop or control looping process. There is three loop control statements are break, continue and goto.

Break Statement: -

This statement is used for terminates the loop. The keyword 'break' use for this. When compiler gets to this statement, it terminates the loop and move to next statement. The switch statement uses it to terminate cases.

Continue Statement: -

Keyword 'continue' is used for continue statement when this keyword arrive to compiler, then compiler skip current statement of loop and start next iteration of the loop.

Goto Statement: -

This statement perform unconditional jump in loop. Keyword 'goto' is use for this with label, and when compiler reaches to goto statement it jump to labels that use before colon symbol [:].

Programmers mostly try to avoid goto statement because it is difficult to trace and control program flow.

Syntax: -

```
goto label;  
...  
...  
...  
Label statement;
```

ARRANGING THE SAME DATA SYSTEMATICALLY: ARRAYS

UNDERSTANDING ARRAYS: -

An array is a type of data structure that can hold multiple values of similar type of data.

DEFINATION: - "Array is a homogeneous data structure that sequentially stores the same type of data and performs different types of operations on these data".

An array is a derived data type for storing similar type of data. The primary data types are integer, float and char. Arrays stores data using this data types because of this an array is known as derived data type.

Data Structure is a concept of the programming language that enables storing and organizing data. Since arrays is also organizes similar data, it is also called a homogeneous and linear data structure. In memory array stores data in contiguous memory location.

It means for example we create an array that store integer value when we see address of that array element it will like 65516, 65518, 65520 because of it is integer array it will took 2 bytes to store integer number. An array look like:

array a[5]	a[0]=1	a[1]=2	a[2]=3	a[3]=4	a[4]=5
------------	--------	--------	--------	--------	--------

memory location	65516	65518	65520	65522	65524
-----------------	-------	-------	-------	-------	-------

Features:-

- Array can store similar data type. Each data item of an array is called array element. For example if data type of an array is integer it will store only integer type of data not other like char and float.
- Array is represent using a subscript written within square brackets '[]'. example Names[]. Value within subscript operator defines size of array. The value within subscript is also called index or subscript variable.
- Computer stores each data element continuously in the memory. As shown above.

ARRAY DECLARATION: -

The declaration of array follow same concept as declaration of variables. While declaring array data type needs to be specified.

Rules for Naming array: -

- The name of array should start with a character or underscore and can include digits.
- The array name should not include any keyword.
- Special symbols should not be used in the array name.

Syntax: -

Data_type array_name [array_size]

Example	Explanation
Int sample[10]	Declares an integer array where the size of the array is 10. The name of the array is sample. It can store any 10 numbers that would be stored from memory location 0 to 9 in the computer memory.
Float rainfall[7]	Declares an float array where the size of the array is 7. The name of the array is rainfall. It can store any 7 numbers that would be stored from memory location 0 to 6 in the computer memory.
Char RN[30]	Declares an character array where the size of the array is 30. The name of the array is RN. It can store any 30 numbers that would be stored from memory location 0 to 29 in the computer memory.
Int person[n]	Declares an integer array where the size of the array is n. The name of the array is person. It can store data of n number of person where the programmer can enter the value of n using scanf(). It would be stored from memory location 0 to (n-1) in the computer memory.

ARRAY INITIALISATION: -

Initialization assigns values to the array elements. A programmer can assign values to arrays at either run time or at compile time.

Initialization at Compile Time: -

Compile time initialization is a type of static binding where values are stored in the program itself. The curly brackets {} are used to initialize the values in the array.

Syntax: -

Data_type array_name [] = {value1, value2,...} or {values}

Example: -

Example	Explanation
Int sample[5] = {11,22,33,44,55}	Declares and initializes an integer array containing 5 values according to the size of the array.
Int sample1[] = {1,2,3,4,5}	Declares and initializes an integer array containing 5 values. In this statement, the size of the array is not provided. Such an expression is executable in the C programming language.
Float height [7] = {5.6,6.1,4.9,5.11,45.3,7.6,8.9}	Declares and initializes a float array containing 7 values according to the size of the array.
Char vowel[5] = {'a','e','i','o','u'}	Declares and initializes a character array containing 5 values according to the size of the array.

Initialization at Run Time: -

Run time initialization is a type of dynamic binding where values are assigned explicitly after execution.

Syntax: -

```
Data_type array_name [size];
Scanf ("%d    %d...%d", &array_name[0], &array_name[1] ,....., &array_name[size-1])
Or
For(i=0;i< size; i++)
{
    Scanf("%d", &array_name[i]);
}
```

Example: -

Example	Explanation
Int s[3]; Scanf ("%d%d%d", &s[0], &s[1], &s[2]);	Here, scanf () function directly reads one by one value in the array's'. In simple words, the first

	value would be store in &s[0], the second value store in &s[1], and the last value in &s[2].
<pre> Int s1[3]; For(i==0;i<3;i++) { Scanf("%d",&s1[i]); } </pre>	<p>Here, for loop is used for initializing values in the array. The loop executes according to the size means 3 times and the scanf () function will read one by one value and store it in continuous memory location. If the size of array is 'n' then the programmer would need to execute the loop such as [i=0;i<n;i++] or [i=0;i<n-1;i++] or [i=1;i<=3;i++].</p>

ACCESSING ARRAY ELEMENTS: -

Array elements are accessed by using index or subscript of the array. Here, index or subscript is a numeric value, which defines the size of the array.

Syntax: - data_type array_name [size] = {value1, value2, value3}

Array_name [0] = value1;

Array_name [0] = value 2;

Example: - int sample [5] = {10, 20, 30, 40, 50}

Sample [0] = 10 [accessing first element]

Sample [1] = 20 [accessing first element]

Sample [2] = 30 [accessing first element]

Sample [3] = 40 [accessing first element]

Sample [4] = 50 [accessing first element]

TWO-DIMENSIONAL ARRAY: -

We see previously one-dimensional array because only index or subscript value is defined. This index is also called dimension and such array called one-dimensional array.

It is possible that array contain more than one dimension or index value. If array contain two dimensions then it called two-dimensional array or 2D-array. The 2D-array is called **MATRIX**.

It is combination of the row and columns. The two indexes of the two-dimensional array are called the rows and column.

Declaration of the 2D Array: -

Syntax: - data_type array_name [row_size] [column_size]

Initialization of the 2D Array: -

We can assign value to array as run time or at compile time.

Syntax: -

```
Data_type array_name [row_size] [column_size] = {value1, value2, ...}  
Or { {column1 value}, {column2 value}, ..., {column n value} }
```

Or

```
Data_type array_name [row_size] [column_size];  
Scanf ("%d %d... %d", &array_name [0] [0], &array_name [0] [1], ..., &array_name  
[row_size-1][column_size-1])
```

Or

```
For (i=0;i< Row_size ;i++)  
{  
    For (i=0; i< column_size; i++)  
    {  
        Scanf ("%d", &array_name[i][j]);  
    }  
}
```

Example: -

```
Int sample [2] [3] = {{11, 45, 67}, {23, 53, 89} };  
Int sample [2] [3] = {{11, 45}, {89}}; // compiler will assign 0 value to remaining elements.  
Int sample [] [2] = {{11, 45}, {89, 56}}; // it is valid to not defined size of row.
```

MULTI-DIMENSIONAL ARRAY (3-D ARRAYS): -

A three-dimensional array is an array of an array, an array you can have two, three, or even more dimensions. As the dimension of an array increases, it can hold more number of data but also the complexity increases.

Declaration of 3-D Array: -

Syntax: -

Data_type array_name [table] [row] [column]; // conceptual syntax of 3-d array.

Data_type array_name [d1] [d2] [d3] [d4] [dn];

Initialization of 3-D Array: -

3-D arrays can be initialized at the time of compilation. An uninitialized 3D array contains garbage values, by default which are not valid for the programming usage.

Example: -

```
int arraylist [3][3][3]    // which hold 3*3*3 =27 elements.
```

You need to know table number, row number and column number to store or access any elements in 3d array.

Example: -

```
Arraylist [1][1][1]    // which will store or access element in array name arraylist in table index 1 in row index in column index 1.
```

Since the memory is allocated in a linear fashion in an array, this will assign values one-by-one to each specified dimension of 3-D array.

CHARACTER ARRAYS

UNDERSTANDING STRINGS:-

It is same as numerical arrays that stores characters. It is group of characters called character array or string. String constant is one-dimensional array, which terminated by null character ['\0'], For example char name[]={ 'a', 't', 'l', 'i', '\0' }.

If string is not terminated by null character then it is not a string but it is only an array of character. In programming, '\0' and '0' are different from each other. Here, '\0' whose ASCII [American Standard Code for Information Interchange] value is 0 whereas '0' is number or digit whose ASCII value is 48.

STRING DECLARATION AND INITIALISATION:-

String Declaration:-

C language does not provide specific data type 'string', but data type char is used to declare string or character array. Name of string follow same rules as variables also declaration of null character '\0' is not necessary, C compiler automatically insert it at end of string.

Syntax:-

```
Char string_name [string_size];
```

Example:-

```
Char person_name [20];
```

String Initialisation:-

Initialisation of string value can be at compile time or run time. The {} initializes the string at compile time and scanf () function assigns values at run time.

Syntax:-

Char stringname [] = {'value1', 'value2', 'value3', 'value4', '\0'} or "values"

Or

Char stringname {};

Scanf ("%s", stringname);

Example	Explanation
Char vowel[6] = {'a', 'e', 'i', 'o', 'u', '\0'};	Declaring and initializing a string for the vowels.
Char consonants[] = {'b','r','t','w'};	Declaring and initializing a string containing 4 characters where the size of the string is not provided. Such expression is executable.
Char string1[20] = "Computer"	Declaring and initializing a string that contains a name "Computer". Here, the size of allocated memory is 20 whereas the initialised string size is 8. Hence, 11 spaces are free so the compiler would allocate NULL here.
Char string2[10]; Scanf ("%s",string2);	The scanf() function directly reads a string from the input device and store it in string2.

OPERATIONS ON STRING:-

Determining length of string, concatenating strings, comparison of two strings, extracting parts of strings, string copy, and others are operations on strings. For these operation codes can be written or use built-in string functions available in the header file "string.h". this file contains many functions that can be used according needs in program.

String Functions	Description	Example
Strlen(stringname) where stringname is a string variable	The function returns the length of the string. The null character is not include in the length of the string.	Strlen("Computer") will return 8.

Strlwr(stringname) where stringname is a string variable	The function converts the given string into lower case letters.	Strlwr("Computer") will return "computer" in output
Strupr(stringname) where string name is a string variable	The function converts the given string into uppercase letters.	Strupr("Computer") will return "COMPUTER" in output.
Strcat(string1,string2) where string1 and string2 are two string variables.	The function concatenates to the string2 with string1.	Strcat("Demo","string") will returns "Demostring" in output.
Strcpy(string1,string2) where string1 and string2 are the two string variables.	The function copies the string2 to the string1 and terminates the string 1 by a Null character.	Strcpy("Demo","string") will returns "string" in output.
Strcmp(string1, string2) where string1 and string2 are the two string variables.	The function compares the string1 and string2 by following lexicographically. The function returns numeric value by comparing strings as follows: String1==string2 then 0 String1<string2 then negative value String1>string2 then positive value	Strcmp("Demo","string") will returnd -1 output.
Strchr(string1,ch) where string1 is a string variable and the ch contains the characters.	The function finds out the characters in the given string and returns a pointer to the first occurrence of the given character in the given string. If the character does not in string then it returns Null.	Strchr("Demo",'X') will returns Null in output. Where as strchr("Demo",'D') will returns Demo in output.
Strstr(string1, string2) where string1 and string2 are the two string variables.	The function finds out the first occurrence of the string2 in string1, which contains string2. If string2 does not occur in string1 then returns null.	Strstr("Demo","Program") will returns Null in output where as strstr("Demo",'m') will return "mo".
Strset(string1,ch) where string1 is a string and the ch contains the characters.	The function replaces the characters in the string1. It returns a pointer to string and stops replacing the character if a null character occurs.	Strset("Demo","D") will replace D into the string "Demo" until a null character occur and retrn "DDDD".
Strrev(string1) where string1 is any string variable	The function reverses the given string.	Strrev("Demo") will returns "omeD" in output.

ARRAYS OF STRINGS:-

An array of string is stores the list of strings. A 2d array stores the array of strings, which is most common application of 2d arrays.

T	O	K	E	N			
K	E	Y	W	O	R	D	
V	A	R	I	A	B	L	E
D	A	T	A	T	Y	P	E
C	O	N	S	T	A	N	T
A	R	R	A	Y			
F	U	N	C	T	I	O	N

An array of strings can be declared and initialised using the two-dimensional arrays as
Char

```
aos[][]={"token","keyword","variable","datatype","constant","array","function"};
```

First dimension defines maximum size of arrays of strings.

GROUP OF STATEMENTS: FUNCTIONS

INTRODUCTION TO FUNCTIONS:-

Technical defination of function is "A function is a block of statements that performs a particular task". Functions can be either pre-defined[built-in] or user-defined. Pre-defined functions are available in programming language and user only need to use it and user-defined functions defined by users according to their requirement.

Main() is a most powerful user-defined function, which required by all programs for execution. Programmer puts all necessary statements within main() function. Each C program is itself a function as it contain a main() function that includes a set of statements for doing any task.

Printf(),scanf() are example of built in function. These built-in functions are also called library functions because these functions stores in header file.

Function provide many advantages to programmers that it reduce number of statements and makes program easy to understand. Function declaration, defination and call are three major parts of function implementation.

FUNCTION DEFINATION:-

Function defination provides common information about function. It has two main components. First is header and second is body.

Function Header:-

It defines name and return type of function.

Function name:- It defines actual name of function. It can be any valid identifiers.

Function type or return type:- It defines the datatype of function means the type of value that function is returning.

Parameter list:- The parameter list defines the list of arguments residing within function and enclosed in parenthesis "()". The comma operator separates the parameters of a function. Each argument has datatype and variable, parameters are optional. They are called parameter because they define information that is passed to a function. They are also classified:-

- Actual parameters appear in function calls.
- Formal parameters appear in function declarations.

Function Body:-

Function body defines the actual statements required for function implementation. Which are enclosed within the curly braces "{}".

- **Local variable declaration:** It defines the variable that are used within the function. Optional part of function body.
- **Executable statements:** It contains the actual statements for function implementation. It can be single or multiple.
- **Return statement:-** It is optional statement of function body. If function return anything then programmer needs it.

Programmer can put function definition anywhere in program either above main() or below main().

Syntax:-

Return_type function_name(parameter list) //function header.

```
{  
    Local variable declaration;  
    Executable statement;  
    ...  
    ... // Function body.  
    ...  
    Return statement;  
}
```

Example:-

Void demo()

```
{  
    Printf("This is sample function");  
}
```

Int demo1(int a,int b)

```
{  
    Printf("This is another line of code");  
    Return 10;  
}
```


FUNCTION DECLARATION:-

Declaration of function is also necessary like declaring variable. The function declaration declares the function and informs the compiler about the structure of the function defining the structure is also known as **function prototyping**.

Declaration contains three elements include function name, type or return type and parameter list(if necessary). Declaration can be above all functions or within main function.

Syntax:-

Return_type function_name(parameter list)

Example:-

```
int demo1(int a,int b);  
void demo();
```

FUNCTION CALL:-

To implement the function definition, it is necessary to call function, which is followed by a list of actual parameters. Whenever a function is called, compiler transfers control to the respective function call. Called function performs particular task and returns values if necessary then compiler return the control back to main function.

Syntax:-

function_name(parameters // if required)

Example:-

```
Demo1(a,b); demo();
```

TYPES OF FUNCTION:-

Function With No Argument And No Return Value:-

This category function does not contain any parameter and does not return any value.

Syntax:-

Void function_name() //declaration

Void function_name()

```
{  
    Local variable declaration;  
    Executable statements;           //definition  
    ...  
    ...  
    ...  
}
```

Function_name() //function call

Function With No Arguments And With Return Value:-

It does not contain any parameter but returns a value.

Syntax:-

```
Data_type Function_name()    //Function declaration
Data_type Function_name()
{
    Local variable declaration;
    Executable statements;    //function definition
    ...
    ...
    ...
    Return statement;
}
Function_name() // function call.
```

Function With Argument And No Return Value:-

It contains parameter but does not return any value.

Syntax:-

```
void function_name(parameter list)    //Function declaration
void function_name(parameter list)
{
    Local variable declaration;
    Executable statements;
    ...
    ...
    ...
}
Function_name(parameter list) //function call
```

Function With Argument And Return Value:-

This category contain both parameter and return value.

Syntax:-

```
Data_type function_name(parameter list) //function declaration
Data_type function_name(parameter list)
{
    Local variable declaration;
    Executable statements;
    ...
    ...
    ...
    Return statement;
```

```
}  
Function_name(parameter list)    //function call
```

Recursion:-

The nested function is a function where one function calls another function or same function and this process is called the nesting of function. C does not support nesting of other function but permits nesting of same function i.e. a function can call itself. This nesting of the function is called recursion.

“Recursion is a process where a function calls itself”. A function that calls itself is called the **recursion function**. Recursion provides many advantages such as it reduces the line of code, unnecessary calling of function, and solves complex problems in an easy way.

Along with advantages, recursion has many disadvantages such as difficult to debug and design, takes much memory in stack, and needs time. In simple programmer face difficulty in tracing the logic of recursive function.

Syntax:-

```
Data_type function_name()  
{  
    Local variable declaration;  
    Executable statements;  
    ...  
    ...  
    ...  
    Function_name();  
    Return statement;  
}
```

Example:-

```
Void sample()  
{  
    Printf("\n recursive function");  
    Sample();  
}
```

PASSING VALUES TO A FUNCTION:-

There is two method for pass value to function during calling of function. **Call by value** and **call by reference** are these method.

Call By Value:-

It is default method for passing values during the calling of a function. This method passes copies of actual values of arguments into the formal argument of function definition.

Example:-

```

Void function1(int x)
{
    X=20;
}
Void main()
{
    Int x=10;
    Printf("\n Before x =%d",x); //it prints x=10;
    Function1(x);                //actual argument in calling function.
    Printf("\n after x =%d",x);   /* It prints x=10 after function calling where value of x
changed (x=20) but here it is not changed*/
}

```

Call By Reference:-

It is second method that copies the address of the actual argument into formal arguments. This method can change the value of argument within the function. If values of the formal arguments change in the called function then it effects the values of the actual arguments in the calling function.

Here, actual values are not safe and by accessing the addresses of the actual arguments, it can be changed from called function. This method is also known as the pass by reference. Address in the calling function where the operator '&' is used. For accessing the value at this address, the dereferencing operator or pointer[*] is used in function definition.

Example:-

```

Void function1(int *x)
{
    *x=20;    //using dereferencing operator.
}
Void main()
{
    Int x=10;
    Printf("\n Before X= %d",x); //It prints x=10
    Function1(&x);                //address of the actual argument in calling function
    Printf("\n After X = %d",x);   //It prints x = 20 after function calling where value of x
                                  changed.
}

```

PASSING ARRAYS AND STRINGS TO A FUNCTION:-

Passing group of values to the function is necessary as it reduces the lines of codes and time. Like other variables, arrays and strings are passed to a function.

Syntax:-

```
Data_type function_name(data_type arrayname[])    //function declaration
Data_type function_name(data_type arrayname[])
{
    Local variable declaration;
    Executable statements;
    ...
    ...                                     //function definition
    ...
    Return statement;
}
Function_name(array_name)    //function call
```

SCOPE OF VARIABLES – STORAGE CLASSES:-

The scope defines a region in which a variable is available whereas the visibility defines an ability for accessing variable from the memory. The time duration in which a variable exists within the memory is called the lifetime of variable. The scope defines the lifetime or visibility of all type of variables.

Types of storage classes:-

- Automatic storage class
- External (or Extern) storage class
- Static storage class
- Register storage class

Automatic Storage Class:-

It defines the scope of variables inside a function. All variables, which are defined within a function, belong to the automatic storage class. By default, all variables are automatic variables. These variable also called local variables, which are local to function. Exiting from the function block automatically destroys these variables. The auto variables stored in main(RAM) memory. The default value of it is garbage value. The keyword “auto” define the auto variable.

External Storage Class:-

It defines the scope of variables outside a function. All variables, which are defined outside function, belong to the external storage class and are called external variables. External variables are also called global variables as they are defined at the top of all functions within a program. The external variable are stored in main(RAM) memory and

their default value is zero. These variable can be defined anywhere in program and can be accessed till the main program.

Keyword “extern” define external variables. External variables are only declared and not initialised. Declaration of extenal variables does not allocate any storage to memory. It should be used when more than one function needs some particular variable.

Example:-

```
Void main()  
{  
    Extern x;  
    Printf(“x=”,x);  
}  
Int x=20;
```

Static Storage Class:-

It defines the scope of variables inside or outside a function but maintain their values until the end of program. The variables belong to the static storage class are called static variables. Keyword “static” defines static variables. These variable can be initialises only once. Static variables can be local to function or global to function. Static variables are stored in ram memory and initialised to zero. It retain the value of the variables between various function calls .

Register Storage Class:-

All variables that belong to register storage class are called register variables. These are local variable stored within the register memory of the microprocessor. Keyword “register” define register variables. The register of memory can store limited variables depend on register size. If the number of variable greater than register then extra register variables are automatically converted to other non-registered variables such as automatic variables. Advantage of these variable is it will be accessed very faster as compared to normal variables beacuae they are stored in register memory rather than main memory.

STORING DIFFERENT DATA TYPES IN SAME MEMORY: STRUCTURES AND UNIONS

DEFINING A STRUCTURE:-

It is user-defined data type that can store different types of data. It is heterogeneous data structure as it contains data types such as int, float, char, double and other in same place. Keyword “struct” define a structure in the program. All members of the structure are enclosed within curly braces need to terminate with a semicolon [;].

Syntax:-

```
Struct structure_name
{
    Data_type structure_member1;
    Data_type structure_member2;
    Data_type structure_member3;
    ...
};
```

Example:-

```
Struct person
{
    Char name[20];
    Int age;
```

```
Char gender[5];  
};
```

STRUCTURE DECLARATION, INITIALISATION AND ACCESSING STRUCTURE MEMBERS:-

Declaration of a Structure Variable:-

The declaration of a structure is different from declaration of other data types. It is user define data type hence, "struct" keyword used for declaring a structure.

Syntax:-

```
Struct structure_name structure_var1, structure_var2, structure_var3;
```

Or

```
Struct struture_name  
{  
... *  
... *  
} structure_var1, structure_var2;
```

Initialisation Of a Structure Variable:-

Structure initialisation stores the values of each member of the structure into curly braces.

Syntax:-

```
Structure_variable = {value of member 1, value of member 2,...};
```

Example:-

```
Person1 = {"Ram",12,'m'};
```

Accessing Structure Members:-

Member access operator or [.] operator use for access the specific element of a structure.

Syntax:-

```
Structure_variable.structure_member
```

Or

```
Structure_variable.structure_member = value
```

APPLICATIONS OF A STRUCTURE:-

A structure is a useful data structure for doing programming efficiently. Structure has many applications.

Array Of Structure:-

To create many structural variable can use array of structure. It will help to reduce code of line.

Syntax:-

```
Struct structure_name structure_variable[size];
```

Example:-

```
Struct person per[20];
```

Nesting Of Structure:-

Structure within structure is called nesting of structure. Outer structure can access all member of the inner structure.

Example:-

```
struct student
{
    Char name[20];
    Int age;
    Struct marks
    {
        Int mark1;
        Int mark2;
    }
}
```

```
Struct student s1;
```

```
Struct marks m1;
```

```
S1.name= value;
```

```
S1.age=value;
```

```
S1.m1.mark1=value;
```

```
S1.m1.mark2=value;
```

Passing Structure To a Function:-

Programmers can pass an entire structure to a function, it pass each member of a structure to function.

Example:-

```
Struct student
{
```

```

        Char name[20];        // structure definition
        Int age;
};
Void show(struct student ss)    //structure in function
{
    Int i;
    For(i=1;i<=n;i++)
    {
        Printf("%s",ss.name);
        Printf("%d",ss.age);
    }
}
Void main()
{
    Int i;
    Struct student s1;    //variable of structure
    Scanf("%s",&s1.name);
    Scanf("%d",&s1.age);
    Show(s1);
}

```

DEFINING UNION:-

Union is also user-defined data type that support storing different kinds of data. It is also heterogeneous data structure, similar to structure but with one major difference is that all members of union use same location for storage where members of structure have own storage. It indicates union can handle one member at a time. Keyword "union" defines union in program. All member of it within curly braces and should terminate with semicolon[;].

Syntax:-

```

Union union_name
{
    Data_type union member1;
    Data_type union member2;
    Data_type union member3;
    ...
};

```

Example:-

Union employee

```
{  
    Char name[20];  
    Int age;  
    Float salary;  
};
```

UNION DECLARATION, INITIALISATION AND ACCESSING UNION MEMBERS:-

Declaration Of The Union Variable:-

Keyword “union” is used to declare a union.

Syntax:-

```
Union union_name union_var1, union_var2,... ;
```

Or

```
Union union_name  
{  
    ... .  
    ... .  
}union_var1, union_var2;
```

Example:-

```
Union employee emp1, emp2;
```

Initialisation Of The Union Variable:-

Syntax:-

```
Union_variable = {value of member1, value of member2, ...};
```

Example:-

```
Emp1 = {"Ram",12,100000.34}
```

Accessing Union Members:-

Syntax:-

```
Union_variable.union_member
```

Or

```
Union_variable.union_member = value
```

Example:-

```
Emp1.name  
Emp1.age=20
```

TYPDEF:-

While using structure or union, programmer needs to use some small names for declaring the variables as they both uses keyword, name, and variable for declaration. In these cases, names becomes too lengthy. The typedef can be used to assign new name to existing datatypes. This feature assigns short names to datatypes.

ADVANTAGES:-

Readability:- As we name the variable for using it structurally in a program, we also name the type to use all the variable of that particular type. It saves the programmer from the bug of incorrect argument ordering.

Portability:- If we want to use same variable name on different system to be of different type, then we keep the typedef in the header file which is controlled by the compiler. This makes typedef portable which can be used from one system to another.

Typedef:- It decreases the complexity while declaring the complex and repeated declarations. **Example** use typedef unsigned long int UNIT64 which is 64 bit wise.

Syntax:-

```
Typedef existing_data_type new_name
```

Example:-

```
Typedef int number; //int n == number n;
```

Or

```
Struct student
{
    Int age;
    Char name[30];
};
Typedef struct student s;
S s1,s2;
```

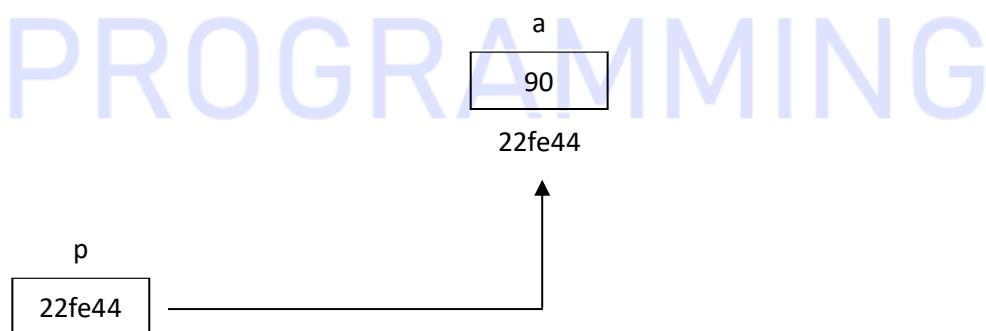
PROGRAMMING

POINTING TO A LOCATION: POINTERS

Pointers enables dynamic memory allocation. Also pointers provide features as increasing execution speed of the program.

INTRODUCTION TO POINTERS:-

Pointer is a variable that holds the address of another variable of the same data type. From this address, the pointer can directly access the value, which is stored in that address.



Features of POINTER:-

- Pointers can directly access memory location, hence, make the execution of a program fast.
- Pointers enable a function return more than one value.
- Pointers enable dynamic memory allocation by directly accessing memory.
- Pointers are used for building complex data structures such as linked lists, stacks, queues, trees and graphs.

POINTER DECLARATION:-

Pointer variables are declared by using the asterisk operator[*]. Naming of pointer variable follow same rules of naming identifiers.

Syntax:-

```
Data_type *pointer_name;
```

Or

```
Data_type* pointer_name;
```

Example:-

Example	Explanation
Int *p OR int* p	Declares an integer pointer or p is pointer variable, which points to int data type.
Float *f	Declares a float pointer
Char *cp	Declares a character pointer
Void *vp	Declares a void pointer, which can points to any kind of data types.

POINTER INITIALISATION AND ACCESSING VALUE USING POINTERS:-

Initialisation Of a Pointer Variable:-

The initialisation of a pointer variable assigns the address of other variables and for this it uses an address of operator[&]. This process is called the referencing as it stores the address or reference of another variable.

Syntax:-

```
Data_type var_name;
```

```
Data_type *ptr_var;
```

```
Ptr_var = &var_name //referencing or storing address of another variable
```

Accessing Values Using a Pointer Variable:-

Pointer variables can also access the value of the specific variables whose address is referenced to that pointer variable. For this the same unary operator[*] is used. This operator is also called the indirection operator. The process of accessing value of the variable whose address is stored in the pointer is called Dereferencing.

Syntax:-

```
Data_type var_name;
```

```
Data_type *ptr_var;
```

```
Ptr_var = &var_name; //storing address of another variable.
```

```
Var_name = *ptr_name //dereferencing or accessing value of pointer variable.
```

Example:-

Example	Explanation
---------	-------------

Int a,b; Int *p; A = 21; P = &a; // referencing B = *p;	Declares two integer variable "a" and "b" and one integer pointer [*p]. the value 21 is assigned to variable "a". The integer pointer is storing the address of variable "a". It means the value of variable "a" is accessible and storing the value of "a" into the variable "b".
Float fa,fb,*fp; Fa = 23.3; Fb = *&fa;	Declares two float variable [fa,fb] and one float variable [fp]. The value 23.3 is assigned to the variable "fa". The line "fb = *&fa" is first storing address the value of variable "fa" and fb is assigned the value 23.3.
Int a, *pa = &a;	Declares a variable "a" and initializing an integer pointer "*pa", which is strong the address of the variable "a".
Int a,b,c,*pd,d,s,t; Pd = &a; D = *pd; Pd = &b; S = *pd; Pd = &c; T = *pd;	Declares some integer variables [a,b,c,d,s,t] and one integer pointer [*pd]. The same pointer is storing the address of three variables and accesing the value of that address one by one.
Int a, *pa, *pb; Pa=&a; Pb=&a;	Declares two integer pointer [*pa,*pb] and one integer variable. Two integer pointers are storing the address of same variable "a".
Int **pa;	Declares a pointer variable, which is pointing to an integer pointer. This is known as pointer to a pointer.

POINTER ARITHMETIC:-

Arithmetic operators with pointers are permitted but they do not support all operator such as multiplication and division. They support increment/decrement, addition and subtraction operators with relational operators for comparison between two pointers.

Syntax:-

Data_type *ptr_var

Ptr_var++ OR ++ptr_var//points to next integer after var.

Ptr_var—OR —ptr_var //points to previous integer after var

Ptr_var + number //points to the next number after var

Ptr_var – number //points to the previous number after var

++*ptr_var //increment var by 1

--*ptr_var //decrement var by 1

Ptr_a < ptr_b //return true if a stored before b;

```
Ptr_a > ptr_b      //return true if a stored after b;
Ptr_a == ptr_b     //return true if ptr_a and ptr_b point to same data element.
Ptr_a != ptr_b     //return true if ptr_a and ptr_b point to different data element but of
                    the same type.
```

Example:-

```
Int *pa,*pb; int a;
```

Valid pointer expression

```
* pa++      //points to next integer after var a.
*pb+2       //increments the value of a by 2.
A=a+*pa     //first adds value of var a to var which stored at address *pa and assigns it to
            the variable a.
A+=*pa      //this expression is equivalent to the above one.
Pa < pb     //it returns true if a is stored before b.
```

Invalid pointer expression:-

```
Pa + pb     //cannot addition of two addresses.
Pa * pb     //cannot multiplication of two addresses.
Pa - pb     //cannot subtraction of two addresses.
Pa / pb     //cannot division of two addresses.
```

POINTER AND ARRAY:-

An array stores a list of similar data and uses indexing for accessing each element. It gets allocated contiguous memory for an array. If array "a" then "a" and "&a[0]" both are same that stores base address (address of first elements of array). Remaining elements stored sequentially after this. Array represents constant pointer whose memory location does not change.

Pointer also follows this concept and stores the address of an element. Difference between them is that array is constant unlike pointer. For printing and storing arrays values loop is used. Array a can get printed by a[i] that prints each element of array. This a[i] converts into a pointer expression that returns value of the array. Hence, a[i] is similar to *(a+i).

Whenever a pointer is used to access an array element then it moves into the computer memory according to the data type. For example there is integer array, and a pointer is used for accessing elements then each loop the pointer will move further by two-memory locations. This is because integer data type takes 2 bytes in memory. For float it will be by four-memory locations.

The "a" and "*a" are equivalents to the "&a[0]" and "a[0]" respectively. In the same way, "a+i" and "*(a+i)" are equivalents to the "&a[i]" and "a[i]" respectively.

ARRAY OF POINTERS:-

For print list of arrays such as array of strings for this 2d array is used which it is necessary to provide column size, a feature to create and implement array of pointers. The array of character pointer supports the string of varying length.

Syntax:-

```
Data_type *ptr_var[size];
```

Example:-

```
Int *pt[10];
```

POINTER AND FUNCTIONS:-

Pointers and function reduce the lines of codes also provide other advantages, it increases the execution speed, there is three way to use it

1. They can pass the pointers to a function as an argument [call by reference], an address of the variable, i.e., the pointer is passed to the function call, which can be further modified in the function definition.
2. The function can return the pointer like other data types. It can be used as – “data_type* function_name()”.
3. You can design pointer to function, which is a function pointer. This function pointer can store the address of other functions and call them using pointers. The statement, “data type (*function_name)()” defines the function pointer.

Example:- void(*fp)(int); fp=&function_name;(*fp)(int);

PROGRAMMING

DYNAMIC MEMORY ALLOCATION AND LINKED LIST

In dynamic memory allocation various functions for allocate and deallocate memory at run time.

UNDERSTANDING DYNAMIC MEMORY ALLOCATION:-

Dynamic Memory Allocation or DMA is the allocation and deallocation of the memory at run time. It helps to define the type of memory, the exact memory required. It concept associated with memory management. OS such as Windows, Unix and other perform it automatically as memory management is one of the tasks of operating system.

C has different types of variables as local, global, and static variable. each of them have own scope, life time, and own memory area. Compiler allocate them in different area, just like local variable are stored in heap. Heap contains free memory of computer. This free memory is used for dynamic memory allocation.

C language does not inherit any technique to dynamically so, it uses some of the functions of `stdlib.h` which is a standard library. This header file contain some predefined functions that implement the dynamic memory allocation and deallocation.

- `Malloc()`
- `Calloc()`
- `Realloc()`
- `Free()`

Malloc):-

This function dynamically allocates a block of memory. It take one argument for define size. It allocates required memory in bytes according to the given size. It supports all data types and returns a void pointer pointing to the first byte of the allocated space.

Syntax:-

```
Void *malloc(size)
```

Example:-

Example	Explanation
Int *ipt; lpt=(int *)malloc(sizeof(int))	It allocates the memory according to the size of integer means 2 bytes.
Int *ipt; lpt=(int *)malloc(40*sizeof(int))	It allocates the 80 bytes in the computer memory according to the size of integer.
Float *ipt; Fpt=(float *)malloc(10*sizeof(float))	It allocates the 40 bytes in the computer memory according to the size of float.
Char *cpt; Cpt=(char *)malloc(15*sizeof(char))	It allocates the 15 bytes in the computer memory according to the size of the character.

Calloc):-

This function dynamically allocates multiple blocks of memory where each one has same size. Each block sets all bytes by zero. Calloc stands for contiguous allocation. The function allocates contiguous space in the memory for the given array. Both calloc() and malloc() function dynamically allocates memory but major difference is calloc() allocates multiple block of memory where as malloc() allocates single block of memory. Calloc() also supports all data types and returns a void pointer.

Syntax:-

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

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

```
#include<stdlib.h>
```

```
Void *calloc (n,element_size)
```

Example	Explanation
Int *ipt; lpt=(int *)calloc(10,sizeof(int))	It will allocate contiguous space in the memory for the array that contains 10 elements. Each element has the size of integer means 2 bytes.
Float *ipt; Fpt = (float *)calloc(20,sizeof(float))	It will allocate contiguous space in the memory for the array that contains 20 elements. Each element has the size of float means 4 bytes.
Char *cpt; Cpt=(char*)calloc(25,szieof(char))	It will allocate contiguous space in the memory for the array that contains 25 elements. Each element has the size of char means 1byte.

Realloc):-

This function reallocates or extends the allocated memory according requirement. Sometimes memory allocated by malloc() and calloc() functions becomes insufficient for working need some extra space then realloc() function reallocates memory.

Syntax:-

Void *realloc(ptr_var,size)

Example	Explanation
Int *ipt; Ipt=(int *)malloc(sizeof(int)) Realloc(ipt,20)	It will reallocate a new memory previously allocated by the malloc() function to size 20 bytes.
Float *ipt; Fpt=(float *)calloc(30,sizeof(float)) Realloc(fpt,50)	It will reallocate a new memory previously allocated by the calloc() function to size 50 bytes.
Char *cpt; Cpt = (char *)malloc(sizeof(char)) Realloc(cpt,20)	It will reallocated a new memory previously allocated by the malloc() function to size 20 bytes.

Free:-

This function release all allocated memory space by malloc() or calloc() functions. For efficient utilization of memory, it is necessary to release the allocated memory dynamically. These function doesn't return value. Compiler automatically releases the memory, which allocated at compile time.

Syntax:-

Free(ptr_var)

Example	Explanation
Int *ipt; Ipt = (int *)malloc(sizeof(int)) Free(ipt)	It will release the allocated memory, which is allocated by the malloc() function.
Char *cpt; Cpt = (char *)calloc(10,sizeof(char)) Free(cpt)	It will release the allocated memory, which is allocated by the calloc() function.

NON-EXECUTABLE SPECIAL LINES: PREPROCESSOR DIRECTIVES

PREPROCESSOR DIRECTIVE:-

C preprocessor is text substitution tool that directs the compiler to perform some pre-processing before starting compilation. It is different from compiler and preprocesses source code. Pre-processor directives are the commands that are used in the preprocessor. Every preprocessing directive starts with a hash sign “#”

Type	Syntax	Explanation
Macro	#define	Macro defines the constant value and supports all data types.
Header file inclusion	#include <filename>	This directive includes the source codes of the program written in the filename and places them at the right place. Many programs of the previous used it as #include<stdio.h>
Conditional compilation	#ifdef #endif #if #else #ifndef	The conditional compilation directive defines a set of commands that instruct the pre-processor whether to include or exclude a chunk of code in the source program before the compilation process with respect to any condition. It generally tests the arithmetic expressions or condition.
Other directives	#undef #pragma	#undef directive undefines a defined macro variable. #pragma calls a function before and after the main function () of a C program.

Conditional preprocessor directives, which instructs the preprocessor whether to select or not piece of code before passing to compiler.

Syntax	Explanation
#ifdef	It checks whether a particular macro is defined or not. If yes, then it returns true otherwise not. After that, the “if clause” statements are included in the program.
#ifndef	It checks whether a particular macro is defined or not, it return true if it is not defined.
#if and #else	It checks whether a compile time condition is true or not. If yes, then includes “if clause” statements otherwise include the “else clause” statements.
#endif	It ends a pre-processor condition.

MACROS:-

A macro is a small piece of code with a suitable macro name. The preprocessor replaces the contents of the macro whenever the macro name occurred in the program. The “#define” keyword represents a macro in a C program. Object-like and function-like are the two types of macros.

Object-like Macro:-

It is a data object, which is replaced by a sequence of token in a program. It is mostly giving meaningful name to any constant.

Syntax:-

```
#define macro_name substitute_text
```

Function-like Macro:-

It is like a function call, which is replaced by a piece of codes in the program. It uses a parenthesis for representing macros. The syntax for defining a function-like macro

Syntax:-

```
#define macro_name() substitute_text
```



PROGRAMMING