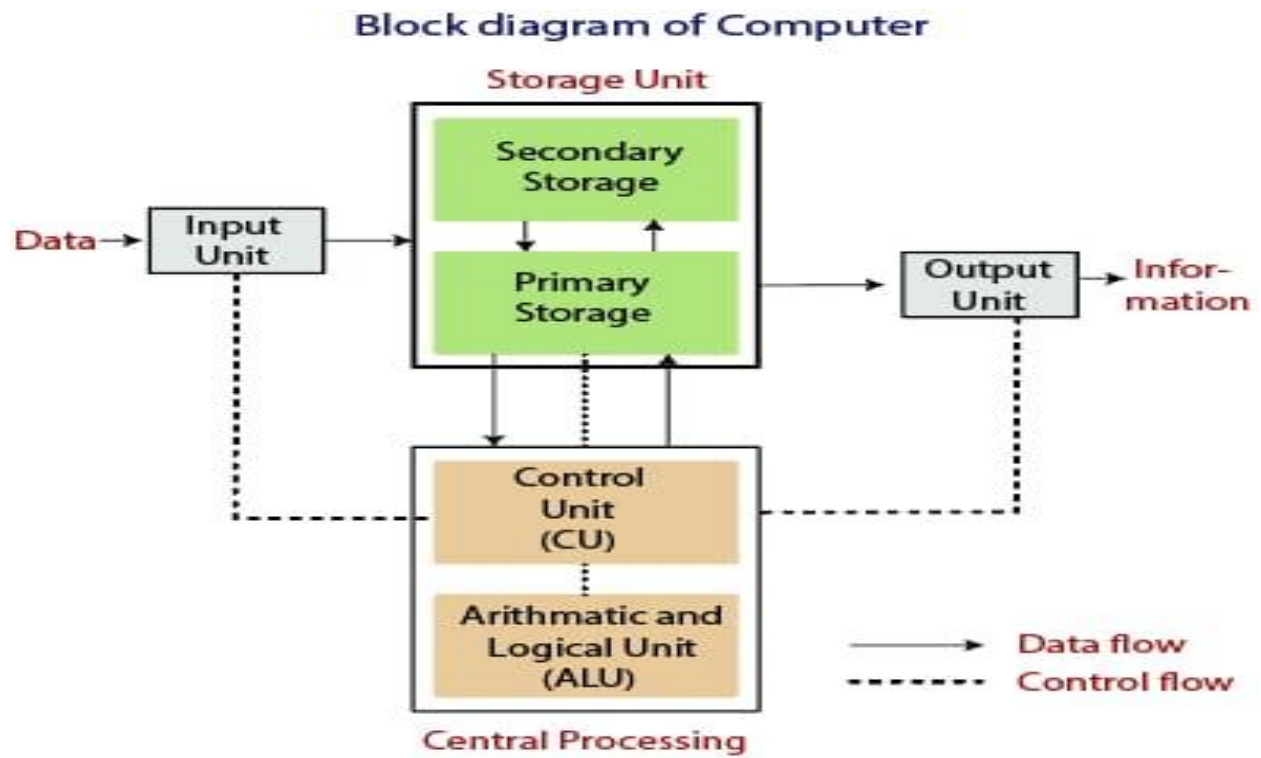


UNIT-1

1. Explain the various functional units of a computer system/Block diagram of a computer system?



The computer performs basically five major operations of functions. These are

- 1) it accepts data or instruction by way of input.
- 2) it stores the data and instructions.
- 3) it can process data as required by the user.
- 4) it gives results in the form of output.
- 5) it controls all operations inside a computer.

1. **Input:** This is the process of entering data and programs into the computer system.
2. **Central Processing Unit (CPU):** The ALU and the CU of a computer system are jointly known as the central processing unit (CPU).
3. **Control Unit (CU):** The process of input, output, processing and storage is performed under the supervision of a unit called 'Control Unit'. It decides when to start receiving data, when to stop it, where to store the data etc. It takes care of step-by-step processing of all operations inside the computer.
4. **Arithmetic Logic Unit (ALU):** The major operations performed by the ALU are addition, subtraction, multiplication, division, logic and comparison etc.
5. **Memory Unit:** Memory unit is used to store data and instructions.
6. **Output:** This is the process of producing results from the data for getting useful information.

2. What are the different types of computing environments?

Computing Environment is a collection of computers which are used to process and exchange information to solve various types of computing problems.

The following are the various types of computing environments.

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

1. Personal Computing Environment

Personal computing is a stand-alone machine. In a personal computing environment, the complete program resides on the stand-alone machine and executed from the same machine. Laptops, mobile devices, printers, scanners and the computer systems we use at home, office are the examples for the personal computing environment.

2. Time-Sharing Computing Environment

The time-sharing computing environment is a stand-alone computer in which a single user can perform multiple operations at a time by using a multitasking operating system. Here the processor time is divided among different tasks and this is called "Time-sharing". For example, a user can listen to music while writing something in a text editor. Windows 95 and later versions of Windows OS, iOS and Linux operating systems are the examples for this computing environment.

3. Client-Server Computing Environment

The client-server environment contains two machines (Client machine and Server machine). These both machines will exchange the information through an application. Here Client is a normal computer like PC, Tablet, Mobile, etc., and Server is a powerful computer which stores huge data and manages the huge amount of file and emails, etc., In this environment, client requests for data and server provides data to the client. In the client-server environment, the communication between client and server is performed using HTTP (Hyper Text Transfer Protocol).

4. Distributed Computing Environment

In the distributed computing environment, the complete functionality of the software is not on a single computer but is distributed among multiple computers. These computers communicate with each other over a network to perform the complete task. In a distributed computing environment, the data is distributed among different systems and that data is logically related to each other.

3. What are the different types of computer languages?

Computer languages are the languages through which the user can communicate with the computer by writing program instructions.

There are three types of computer languages

1. Machine level language
2. Assembly level language
3. High level language

1. Machine level language:

It's nothing but set of instructions given to the compiler in the forms of 0's and 1's. Machine language consists of 0's and 1's. This is the language of computers. Various operations can be performed by the combination of 0's and 1's. It is also called binary language.

Advantages:

1. Program execution is very fast.
2. Since the computer can understand and execute machine language, the translator is not required.

Disadvantages:

1. Difficult to remember the machine instructions.
2. Difficult to read and write machine language programs.
3. Very difficult to debug, correct and modify.
4. They are machine dependent and not portable.
5. It is unstructured language.

2. Assembly level language:

It is also called as Symbolic language which is nothing but the set of instruction given to the computer in the form of symbolic names. The symbolic names are also called mnemonics.

Example:-

ADD R1,R2 (adds the values of R1 and R2)

MUL R1,R2 (multiplies the values of R1 and R2)

Advantages:

1. Easy to remember the symbolic names and there is no need to remember the machine code.
2. Program understanding, modification is relatively easier when compared to machine language.
3. The execution speed of a program written in assembly language is same as that of the equivalent program written in machine language.

Disadvantages:

1. There is a need for translating the assembly language program into machine language program.
2. Like machine language, the assembly language is also machine dependent.
3. It is unstructured language.
4. Difficult to understand and debug when compared to high level language.

3. High level language:

A high level language is one, which is written using symbols and words just like English language. The high level languages enable the programmer to write machine independent code.

Advantages:

1. Easy to understand.
2. Easy to read write and modify.
3. The code is very compact.
4. It is structured language.
5. They are machine independent programs.

Disadvantages:

1. Take more time to execute when compared with machine language or assembly language.
2. Translator is required to convert the programs written in high-level language to machine language.

4. What is an algorithm? Explain its properties with an example?Algorithm:

An algorithm is a step by step process to solve a problem.

Properties of an Algorithm:

Every algorithm should have the following 5 properties.

1. Input: An algorithm may take one or more inputs.
2. Output: An algorithm should produce at least one output.
3. Finiteness: An algorithm should end after a fixed number of steps.
4. Definiteness: Each instruction of an algorithm must be clear and well defined.
5. Effectiveness: The operations must be simple and completed within a finite time.

Example: Algorithm to find the largest number among 3 numbers.

Step 1: Start

Step 2: Declare the variables A, B, C

Step 3: Read/Input the values of A, B, C

Step 4: Check if (A > B) then goto step 5, otherwise goto step 6

Step 5: Check if (A > C) then goto step 7, otherwise goto step 9

Step 6: Check if (B > C) then goto step 8, otherwise goto step 9

Step 7: Output "A is the largest", then goto step 10

Step 8: Output "B is the largest", then goto step 10

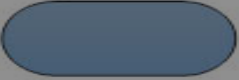

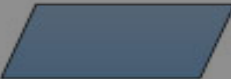
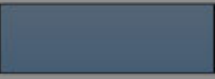

Step 9: Output "C is the largest", then goto step 10

Step 10: Stop

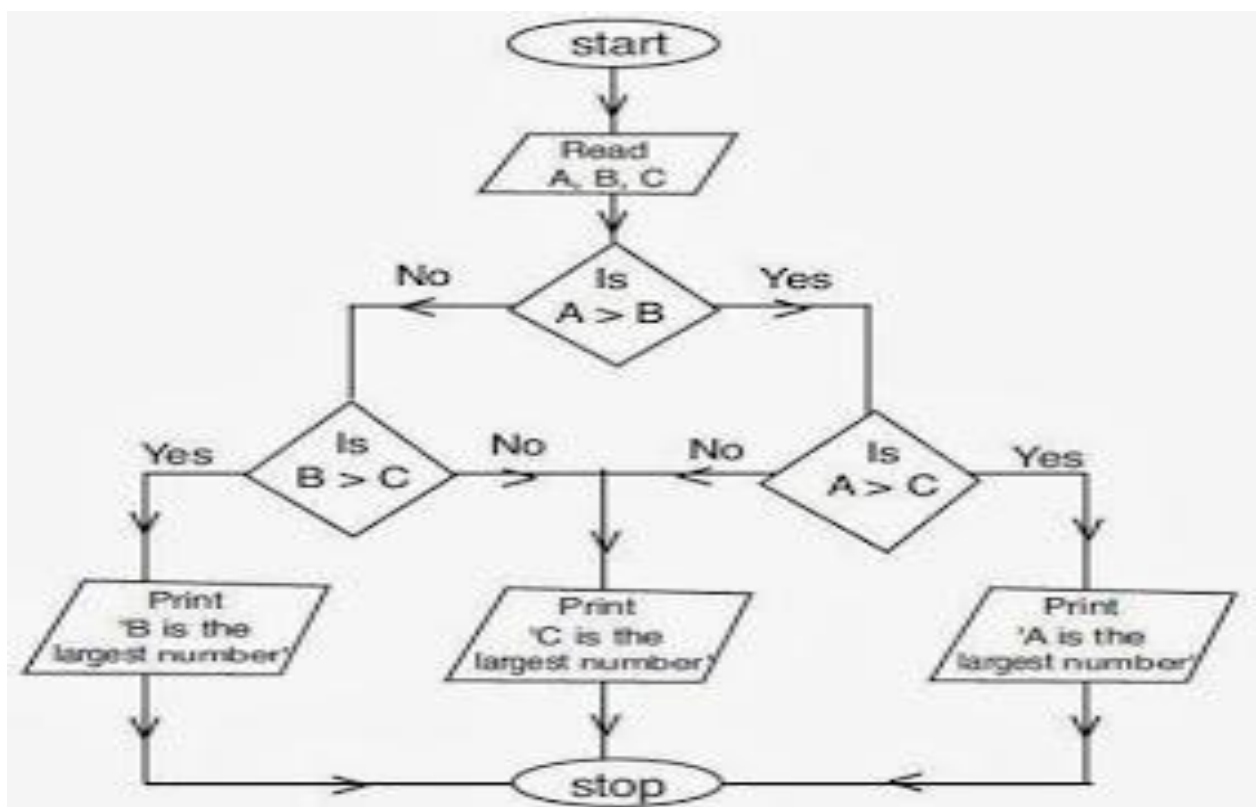
5. What is a flowchart? Explain the symbols used in a flowchart with an example?Flowchart:

A graphical/pictorial representation of an algorithm is called flowchart.

The following symbols are used to construct a flow chart.

Symbol	Name	Function
	Start/end	An oval represents a start or end point
	Arrows	A line is a connector that shows relationships between the representative shapes
	Input/Output	A parallelogram represents input or output
	Process	A rectangle represents a process
	Decision	A diamond indicates a decision

Example: Flowchart to find the largest number among 3 numbers.



6. What are the different steps to be followed in the program development/How to creating and running a C program?

A C program has to pass through many phases for its successful execution and to achieve the desired output.

The various steps involved in the program development are as follows:

1. Editing phase
2. Compilation phase

3. Linking phase
4. Execution phase

1. Editing Phase:

In the editing phase, the C program is entered into a file through a text editor.

The file is saved on the disk with an extension of 'C'. Corrections in the program at later stages are done through these editors. Once the program has been written, it requires to be translated into machine language.

2. Compilation Phase:

This phase is carried out by a program called as compiler. Compiler translates the source code into the object code. The compilation phase cannot proceed successfully until and unless the source code is error-free. The compiler generates messages if it encounters syntactic errors in the source code. The error-free source code is translated into object code and stored in a separate file with an extension '.obj'.

3. Linking Phase:

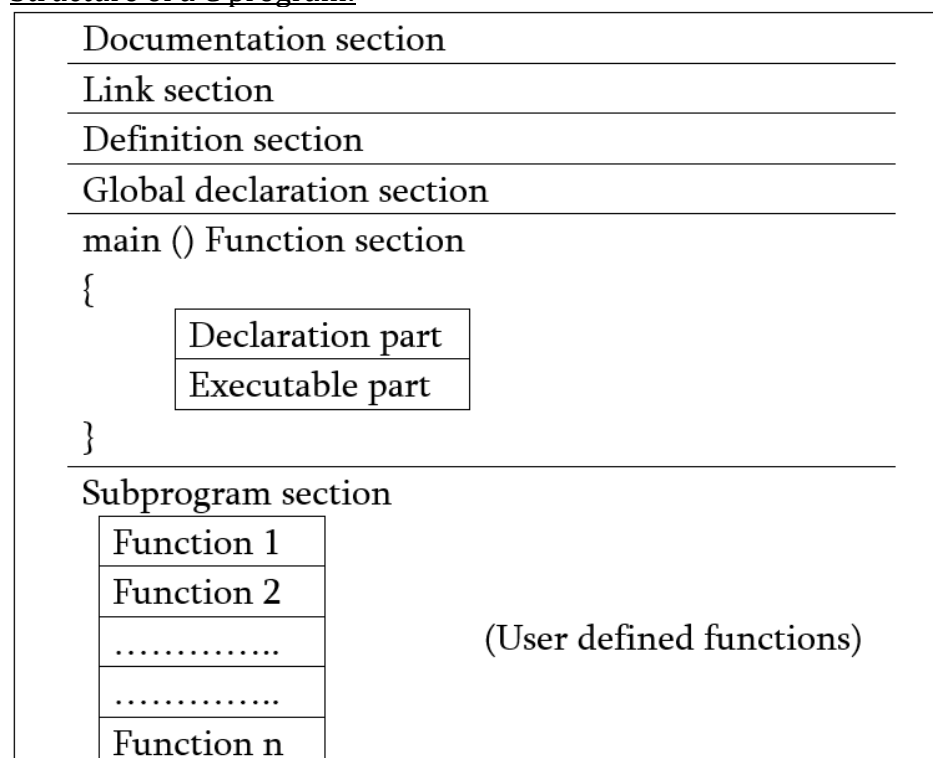
In this phase, the linker links the files and functions with the object code under execution. For example, if a user uses a 'printf' function in his\her program, the linker links the user programs, object code with the object code of the 'printf' function. The result of this linking process produces an executable file with an extension of '.exe'. Now the object code is ready for next phase.

4. Execution Phase:

In this phase, the executable object code is loaded into the memory and the program execution begins. We may encounter errors during the execution phase even though compilation phase is successful. The errors may be run-time errors and logical errors.

7. Explain the structure of a C program?

Structure of a C program:



1. Documentation section:

It is a set of comment lines that describes the name of the program.

Examples: // Write a C program to find the sum of 2 Numbers

/* Write a C program to find the simple interest*/

2.Link section:

It tells the compiler to link functions from the system library.

Examples:

#include<stdio.h>

#include<conio.h>

#include<math.h>

#include is a pre-processor directive command.

stdio is a standard input & output.

.h is an extension of header file.

3. Definition section:

It defines all the symbolic constants and assigns them some value.

Example: #define PI 3.14

Here #define is a pre-processor directive command which tells the compiler whenever PI is found in the program replace it with 3.14.

4.Global declaration section:

There are some variables used more than one function such variables are called global variables and are declared outside of all the functions.

5.main() function section:

Every C program should contain one main() function.

Every C program execution starts with main().

This section contains two parts.

1.Declaration part:

This part declares all the variables which are used in the executable part.

Examples:

int a, b, c;

float x, y, z;

char name, branch;

2. Executable part:

This part contains atleast one statement.

All the statements in the declaration and executable parts should be end with semicolon (;).

The program execution begins at the opening brace and ends with closing brace.

6. Subprogram section/user defined function section:

It contains all the user defined functions that are called in the main() function section.

All the user defined functions are placed before or after the main() function section.

8. Define C tokens and explain in detail?

C Tokens:

The basic or smallest units in C program are called C Tokens. They are 6 types of tokens in C language. C programs are written using these tokens and syntax of the language.

1. Keywords/Reserved words
2. Identifiers

3. Constants/Literals
4. Strings
5. Special symbols
6. Operators

1. Keywords: Keywords are fixed and predefined meaning in C language. There are 32-Keywords.

32 Keywords in C Programming Language

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

Rules for declaring keywords:-

1. Keywords must be written in the lower case letters.
2. Keywords should not be declared as variables or identifiers.

2. Identifiers: Identifiers are names given to the variables, functions and arrays defined by the user. Identifier is a sequence of letters, digits and one special character i.e., _ (Underscore).

Ex:- a, b, abc, abc_123, a123, xyz etc.

Rules for declaring Identifiers:-

1. Identifiers must be start with a letter or _ (Underscore).
2. Identifiers should not start with a digit.
3. No special symbols other than _ (Underscore) are allowed within the identifiers.
4. Identifiers are case sensitive i.e., the uppercase and lowercase letters are different.
5. Identifiers should not be declared as keywords.
6. Identifiers can be a length of first 31 characters.

3.Constant/Literals:-

Constants are the fixed values that do not change during the execution of a program.

Constant are divided into two types.

1. Numeric constants
2. Non-Numeric /Character constants

1.Numeric Constants: There are two types of Numeric constants

1. Integer constants
2. Real or floating point constants

1.Integer constants:- It is a sequence of digits without decimal points.

Ex: - 10, 20, 30 etc.

Rules for declaring Integer constants:-

1. It must contain at least one digit.
2. It should not contain any decimal point.
3. It could be either positive integer or negative integer constant.
4. If no sign precedes an integer constant, it is assumed to be positive.

2.Real/Floating point constants: It is a sequence of digits with decimal points.

Ex:- 10.32 , 2.02 etc.

Rules for declaring Real constants:-

1. It must contain at least one digit.
2. It should contain decimal point.
3. It could be either positive or negative real constant.
4. If no sign precedes a real constant, it is assumed to be positive.

2.Non-Numeric Constants/Character constant:- There are two types of character constants

1. Single character constants
2. String character constants

1.Single character constants:- Any single letter or digit enclosed within single quotation marks is called single character constant.

Ex:- 'A','5'

2.String character constant:- String is a group of characters enclosed within the double quotation marks is called String character constants.

Ex:- "MRCET", "CSEA", "AIML"

5.Special symbol:- { }, (), [], \$, # , @ are the special symbols.

6.Operators:- Operator is a symbol which specifies an operation to be performed on the operands.

Ex:-Arithmetic operators, Logical operators, Relational operators, Increment and Decrement operators etc.

9. Write about variables and its syntax and rules?**Variables:**

1. Variable is a data name which can be used to store a data value.
2. Variables can be changed because it takes different values at different times during the execution of a program.

Declaration/Syntax of a variable:-

datatype variable1,variable 2, . . . ,variable n;

Ex1:- int a, b, c;

Ex2:- float x, y,z;

Assigning values to variables:-Values can be assigned to a variable using assignment operator(=) in C language.

Declaration/Syntax:-

datatype variable =constant;

Ex1:- int a=10,b=34,c=43;

Ex2:- float x=10.3,y=34.15;

Rules for declaring variables:-

1. Variable is a sequence of letters, digits,special character ' _ ' (Underscore).
2. Variables must be start with a letter or _ (Underscore).
3. Variables should not start with a digit.
4. No special symbols other than _ (Underscore) is allowed within the variables.
5. Variables can be a length of first 31 characters.

6. Variables are case sensitive i.e., the upper case and lower case letters are different.
7. Variables should not be declared as keywords.

10. Explain all primary/fundamental/standard data types in C?

Data types:

1. Data type indicates what kind of data can be stored in the variables and identifiers.
2. It can be used for storage representations.
3. Data types are fixed and predefined meaning in C language.
4. Data types can be divided into four types: -
 - i) Primary / fundamental data types
Ex: - int, short int, long int, float, double, long double, char.
 - ii) Derived data types
Ex: - arrays, pointers, functions, strings, structures, unions.
 - iii) User defined data types
Ex: - typedef, enum.
 - iv) Empty data type
Ex: - void.

I.Primary/Fundamental data types: -

There are three types:-

1. Integer Data types
2. Character Data types
3. Real/Floating point Data types

1.Integer Data type: -

It can be used to represent/store integer constants.

There are 3 types.

1.int

2.short int

3.long int

1.int: -

1. It is indicated by the keyword 'int'.
2. The control string of 'int' is %d.
3. It occupies 2 bytes of memory location i.e., 16bits.
4. The range of int is -32768 to 32767.

Ex: - int x,y;
scanf ("%d%d",&x,&y);

2.short int: -

1. It is indicated by the keyword 'short'.
2. When the given value is less than the int range then short int is used.
3. The control string of short int is %d.
4. It occupies 1 byte of memory location i.e., 8bits.
5. The range of short int is -128 to 127.

Ex: - short int x,y; or short x,y;

scanf ("%d%d",&x,&y);

3.long int: -

- 1.It is indicated by the keyword 'long'.
- 2.When the given value is greater than the 'int' range then long int is used
- 3.The control string of long int is "%ld"
- 4.It occupies 4 bytes of memory location i.e., 32bits.
- 5.The range of long int is -2,147,483,648 to 2,147,483,647.

Ex: - long int x,y; or long x,y;
scanf("%ld%ld",&x,&y);

2.Character Data types: -

It can be used to represent/store either single character constants or string character constants.

- 1.It is indicated by the keyword 'char'.
- 2The control string char is '%c' for single character constants and '%s' for string character constants.
- 3.It occupies 1byte of memory location i.e., 8bits.
- 4.The range of char is -128 to 127.

Ex1: char x,y;
scanf("%c%c",&x,&y);
Ex2: char name, college;
scanf("%s%s",&name,&college);

3.Real/Floating point data types: -

It can be used to represent store Real/Floating point constants.

There are 3 types.

1.float

2.double

3.long double

1.float: -

- 1) It is indicated by the keyword 'float'.
- 2) The control string of float is "%f".
- 3) It occupies 4bytes of memory location. i.e., 32bits.
- 4) The range of float is 3.4E-38 to 3.4E+38.

Ex: - float a,b;
scanf("%f%f",&a,&b);

2.double: -

- 1) It is indicated by the keyword 'double'.
- 2) When the given value is greater than the float range then double is used.
- 3) The control string of double is "%lf".
- 4) It occupies 8bytes of memory location i.e., 64bits.
- 5) The range of double is 1.7E-308 to 1.7E+308.

Ex: - double a,b;
scanf("%lf%lf",&a,&b);

3.long double: -

- 1) When the given value is greater than the double range then long double is used.
- 2) The control string of long double is "%Lf".
- 3) It occupies 10 bytes of the memory location i.e., 80bits.
- 4) The range of long double is 3.4E-4932 to 1.1E+4932.

Ex: long double a,b;
scanf("%Lf%Lf",&a,&b);

11. Explain different types of operators in C?

Operator is a symbol which indicates an operation to be performed between the operands (Variables or Values).

- 1.Arithmetic operators: +, -, *, /, %
- 2.Relational operators: >, <, <=, >=, ==, !=
- 3.Logical operators: &&, ||, !
- 4.Assignment operators: =

5. Conditional/Ternary operator: `?:`

6. Increment and Decrement operators: `++`, `--`

7. Bitwise operators: `&`, `|`, `^`, `<<`, `>>`, `~`

8. `sizeof()` operator: `sizeof()`

9. Special operators: Comma operator(`,`), dot/member operator(`.`), address operator(`&`), value at address operator (`*`), arrow/selection operator(`->`)

1. Arithmetic operator:-

1. Arithmetic operators are used to perform basic arithmetic operations between the operands.
2. There are 5 types of Arithmetic operators in C language they are `+`, `-`, `*`, `/`, `%`.
3. Division operator(`/`) gives the result in quotient.
4. Modulus operator(`%`) gives the result in remainder.
5. `%` operator cannot be worked with the real constants.

2. Relational operators:-

1. Relational operators are used to compare the relation between the operands.
2. There are 6 types of relational operators in C language they are `<`, `>`, `<=`, `>=`, `!=`, `==`.
3. Relational operator gives the result either in 1 (true) or 0 (false).

3. Logical operators:-

1. Logical operators are used to combine two or more relational conditions.
2. Logical operators gives the result either in 1 (true) or 0 (false).
3. There are 3 types of Logical operators in c language
 1. Logical AND (`&&`).
 2. Logical OR (`||`).
 3. Logical NOT (`!`).

4. sizeof operator:- This operator can be used to find the number of bytes occupied by a variable or data type.

Syntax:- `sizeof(operand);`

5. Assignment operator:- Values can be assigned to a variables using assignment operator in c language. The symbol of assignment operator is `'='`.

Assignment operator supports 5 short hand assignment operators in C language they are `+=`, `-=`, `*=`, `/=`, `%=`

6. Conditional operator/Ternary operator:-

In C language `'?'` and `':'` are called conditional operator or ternary operators in C language.

Syntax: `Condition ? TRUE Part : FALSE Part;`

Here first the condition is checked, if it is true then TRUE Part is executed otherwise FALSE Part is executed.

7. Increment and Decrement operators:-

1. These operators are used to increment or decrement the value of the variable by 1.
2. `'++'` is the increment operator in C language.

3. '--' is the decrement operator in C language.
4. There are 4 types of operators in C language.
 1. Pre-increment operator
 2. Post-increment operator
 3. Pre-decrement operator
 4. Post-decrement operator

1. Pre-increment operator:

1. ++ operator is placed before the operand is called pre-increment operator. Ex: ++x, ++a
2. Pre-increment operator specifies first increment the value of the variable by 1 and then assigns the incremented value to other variable.

2. Post-increment operator:

1. ++ operator is placed after the operand is called post-increment operator. Ex: x++, a++
2. Post-increment operator specifies first assigns the value of the variable to other variable and then increments the value of the variable by 1.

3. Pre-decrement operator:

1. -- operator is placed before the operand is called pre-decrement operator. Ex: --x, --a
2. Pre-decrement operator specifies first decrement the value of the variable by 1 and then assigns the decremented value to other variable.

4. Post-decrement operator:

1. -- operator is placed after the operand is called post-decrement operator. Ex: x--, a--
2. Post-decrement operator specifies first assigns the value of the variable to other variable and then decrements the value of the variable by 1.

8. Bitwise operators: The data stored in a computer memory as a sequence of 0's and 1's. There are some operators which work with 0's and 1's are called bitwise operators. Bitwise operators cannot be worked with real constants.

There are 6 types of operators in C language.

1. Bitwise AND (&)
2. Bitwise OR (|)
3. Bitwise X-OR (^)
4. Bitwise left shift (<<)
5. Bitwise right shift (>>)
6. Bitwise 1's complement (~)

1. Bitwise AND (&): The result of the bitwise AND is 1 when both the bits are 1 otherwise 0.

2. Bitwise OR (|): The result of the bitwise OR is 0 when both the bits are 0 otherwise 1.

3. Bitwise X-OR (^): The result of the bitwise X-OR is 1 when one bit is 0 and other bit is 1 otherwise 0.

4. Bitwise left shift (<<): This operator can be used to shift the bit positions to the left by 'n' positions.

5. Bitwise right shift (>>): This operator can be used to shift the bit positions to the right by 'n' positions.

6.Bitwise 1's complement (~):This operator can be used to reverse the bit i.e., it changes from 0 to 1 and 1 to 0.

12. Write about operator precedence and associativity of operators with an example expression?

Operator Precedence

Operator precedence is used to determine the order of operators evaluated in an expression. In c programming language every operator has precedence (priority). When there is more than one operator in an expression the operator with higher precedence is evaluated first and the operator with the least precedence is evaluated last. This rule of priority of operators is called 'operator precedence'.

Operator Associativity

Operator associativity is used to determine the order of operators with equal precedence evaluated in an expression. In the c programming language, when an expression contains multiple operators with equal precedence, we use associativity to determine the order of evaluation of those operators.

The following table shows the precedence and associativity of operators:

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

In the C programming language, an expression is evaluated based on the operator precedence and associativity. When there are multiple operators in an expression, they are evaluated according to their precedence and associativity. The operator with higher precedence is evaluated first and the operator with the least precedence is evaluated last.

To understand expression evaluation in c, let us consider the following simple example expression.

10 + 4 * 3 / 2

In the above expression, there are three operators **+**, ***** and **/**. Among these three operators, both multiplication and division have the same higher precedence and addition has lower precedence. So, according to the operator precedence both multiplication and division are evaluated first and then the addition is evaluated. As multiplication and division have the same precedence they are evaluated based on the associativity. Here, the associativity of multiplication and division is **left to right**. So, multiplication is performed first, then division and finally addition. So, the above expression is evaluated in the order of *** / and +**. It is evaluated as follows...

4	*	3	====>	12
12	/	2	====>	6
10	+	6	====>	16

The expression is evaluated to **16**.

13. Write about type conversion and type casting in C?

In a c programming language, the data conversion is performed in two different methods as follows.

1. Type Conversion
2. Type Casting

1. Type Conversion

The type conversion is the process of converting a data value from one data type to another data type automatically by the compiler. Sometimes type conversion is also called implicit type conversion. The implicit type conversion is automatically performed by the compiler.

For example, in c programming language, when we assign an integer value to a float variable the integer value automatically gets converted to float value by adding decimal value 0. And when a float value is assigned to an integer variable the float value automatically gets converted to an integer value by removing the decimal value. To understand more about type conversion observe the following...

```
int i=10;
float x=15.5;
char ch='A';
```

`i = x ;` =====> x value 15.5 is converted as 15 and assigned to variable i

`x = i ;` =====> Here i value 10 is converted as 10.000000 and assigned to variable x

`i = ch ;` =====> Here the ASCII value of A (65) is assigned to i

2. Type Casting

It is also called explicit type conversion and it is used to convert from one data type to another data type. To convert data from one type to another, we specify the new type in parentheses before the value we want converted.

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

```
int a;
```

(float) a;

14. Write about typedef, enum in C language?

typedef(type definition)

It is a user defined datatype and typedef is used to create a new data type for the existing data type.

Syntax:

```
typedef datatype newname;
```

In the above syntax datatype may be any basic datatype such as int,float,char etc. newname is user defined name for that data type.

Example:

```
typedef int MRCET;
```

In the above example MRCET is user defined data type for int data type so that we can use MRCET in place of int in a program as follows

```
MRCET a,b,c;
```

enum(enumerated datatype):

It is a user defined datatype and enum is used to create a new data type and declaring enumeration types.

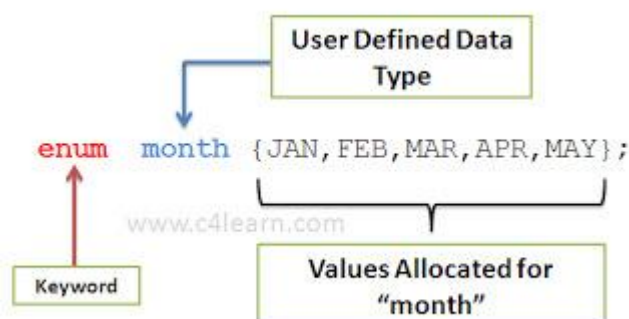
Syntax:

```
enum enumerartiontype{identifier1,identifier2,.....,identifier n};
```

Example: Consider 12 months of a year.

```
enum month{jan,feb,mar,.....dec};
```

In the above example month is a user defined data type or enumeration type and jan,feb,.....dec are the identifiers and their values starts from zero(0) so jan refers to 0,feb refers to 1,mar refers to 2,...dec refers to 11.



15. Explain conditional statements/decision making/selection statements in C?

Conditional/Decision making statements.

There are 4 types of Conditional/Decision making statements in C language.

They are

1. if
2. if else
3. nested if else

4. switch statement

1.if statement: It can be used to execute a single statement or a group of statements based on the condition.

It is a one way branching statement in C language.

Syntax:

```
if (condition)
```

```
{
```

```
Statements;
```

```
}
```

```
next statement;
```

Operation: First the condition is checked if it is true then the statements will be executed and then control is transferred to the next statement in the program.

if the condition is false, control skips the statements and then control is transferred to the next statement in the program.

Ex: Write a C program to check whether the given number is greater than 25.

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

```
int main()
```

```
{
```

```
int n;
```

```
printf("Enter any number\n");
```

```
scanf("%d",&n);
```

```
if (n>25)
```

```
printf("The given number is greater than 25");
```

```
}
```

2.if else statement: It can be used to execute a single statement or a group of statements based on the condition.

It is a two way branching statement in C language.

Syntax:

```
if (condition)
```

```
{
```

```
statement1;
```

```
}
```

```
else
```

```
{
```

```
statement2;
```

```
}
```

```
next statement;
```

Operation: First the condition is checked if the condition is true then statement1 will be executed and it skips statement2 and then control is transferred to the next statement in the program.

if the condition is false then statement1 is skipped and statement2 will be executed and then control is transferred to the next statement in the program.

Ex: Write a C program to check whether the given number is even or odd.

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

```
int main()
```

```
{
```

```
int n;
```

```
printf("Enter any number\n");
```

```
scanf("%d",&n);
```

```
if (n%2==0)
printf("The given number is even");
else
printf("The given number is odd");
}
```

3.nested if else statement: If we want to check more than one condition then nested if else is used.

It is multi way branching statement in C language.

Syntax:

```
if (condition1)
{
if (condition2)
{
statement1;
}
else
{
statement2;
}
}
else
{
statement3;
}
next statement;
```

Operation: First the condition1 is checked if it is false then statement3 will be executed and then control is transferred to the next statement in the program.

If the condition1 is true then condition2 is checked, if it is true then statement1 will be executed and then control is transferred to the next statement in the program.

If the condition2 is false then statement2 will be executed and then control is transferred to the next statement in the program.

Ex: Write a C program to find the largest number among three numbers using nested if else.

```
#include<stdio.h>
int main()
{
int a,b,c;
printf("Enter the values of a,b,c\n");
scanf("%d%d%d",&a,&b,&c);
if (a>b)
{
if (a>c)
printf("a is largest number");
else
printf("c is largest number");
}
else
{
if(b>c)
printf("b is largest number");
else
```

```
printf("c is largest number");
}
}
```

4.switch statement: If we want to select one statement from more number of statements then switch statement is used.

It is a multi way branching statement in C language.

Syntax:

```
switch(expression)
{
case value1:block1;
break;
case value2:block2;
break;
-----
-----
case valuen:blockn;
break;
default :default block;
break;
}
next statement;
```

Operation: First the expression value (integer constant/character constant) is compared with all the case values in the switch. if it is matched with any case value then the particular case block will be executed and then control is transferred to the next statement in the program.

If the expression value is not matched with any case value then default block will be executed and then control is transferred to the next statement in the program.

Ex: Write a C program to perform all arithmetic operations between 2 operands using switch statement.

```
#include<stdio.h>
int main()
{
int x,y,sum,sub,mul,div,mod,choice;
printf("Enter the value of x and y\n");
scanf("%d%d",&x,&y);
printf("Enter your choice\n");
printf("1.Addition\n2.Subtraction\n3.Product\n4.Quotient\n5.Remainder\n");
scanf("%d",&choice);
switch(choice)
{
case 1:sum=x+y;
printf("The sum of x and y is %d",sum);
break;
case 2:sub=x-y;
printf("The sub of x and y is %d",sub);
break;
case 3:mul=x*y;
printf("The mul of x and y is %d",mul);
break;
case 4:div=x/y;
printf("The div of x and y is %d",div);
```

```

break;
case 5:mod=x%y;
printf("The mod of x and y is %d",mod);
break;
default :printf("Invalid Choice");
break;
}
}

```

16. Explain loops/repetition/iterative statements in C?

Loops or Repetition statements:-

There are 3 types of Loops/Repetition statements in C language.

They are

1. while
2. do while
3. for

1.while loop statement: It is used when a group of statements are executed repeatedly until the specified condition is true.

It is also called entry controlled loop.

The minimum number of execution takes place in while loop is 0.

Syntax:

```

while(condition)
{
body of while loop;
}
next statement;

```

Operation: First the condition is checked if the condition is true then control enters into the body of while loop to execute the statements repeatedly until the specified condition is true.

if the condition is false, the body of while loop is skipped and control comes out of the loop and continues with the next statement in the program.

Ex: Write a C program to print the numbers from 1 to 5 using while.

```

#include<stdio.h>
int main()
{
int i=1;
printf("The numbers from 1 to 5 are\n");
while(i<=5)
{
printf("%d\n",i);
i++;
}
}

```

2.do while loop statement:-

It is used when a group of statements are executed repeatedly until the specified condition is true.

It is also called exit controlled loop.

The minimum number of execution takes place in do while loop is 1.

Syntax:-

```

do

```

```
{  
body of do while loop;  
}  
while(condition);  
next statement;
```

In do-while loop condition should be end with semicolon.

Operation: In do-while loop the condition is checked at the end i.e., the control first enters into the body of do while loop to execute the statements. After the execution of statements it checks for the condition. if the condition is true then the control again enters into the body of do while loop to execute the statements repeatedly until the specified condition is true.

if the condition is false then control continues with the next statement in the program.

Ex: Write a C program to print the numbers from 1 to 5 using do while.

```
#include<stdio.h>  
int main()  
{  
int i=1;  
printf("The numbers from 1 to 5 are\n");  
do  
{  
printf("%d\n",i);  
i++;  
}  
while(i<=5);  
}
```

3.for loop statement:- It is used when a group of statements are executed repeatedly until the specified condition is true.

It is also called entry controlled loop.

The minimum number of execution takes place in for loop is 0.

Syntax:-

```
for (initialization;condition;increment/decrement )  
{  
body of for loop;  
}  
next statement;
```

Operation: First initial value will be assigned. Next condition is checked if the condition is true then control enters into the body of for loop to execute the statements. After the execution of statements the initial value will be incremented/decremented. After initial value will be incremented/decremented the control again checks for the condition. If the condition is true then the control is again enters into the body of for loop to execute the statements repeatedly until the specified condition is true.

if the condition is false, the body of for loop is skipped and control comes out of the loop and continues with the next statement in the program.

Ex: Write a C program to print the numbers from 1 to 5 using for.

```
#include<stdio.h>  
int main()  
{  
int i;  
printf("The numbers from 1 to 5 are\n");  
for(i=1;i<=5;i++)
```

```
{
printf("%d\n",i);
}
}
```

17. Explain unconditional statements/jump statements in C?

Unconditional statements:- Normal flow of control can be transferred from one place to another place in the program. This can be done by using the following unconditional statements in the C language.

There are 3 types of Unconditional statements in C language.

They are

1. goto
2. break
3. continue

1.goto statement:- goto is an unconditional statement used to transfer the control from one statement to another statement in the program.

Syntax:-

goto label;

label:

Statements;

Ex: Write a C program to check whether the given number is even or odd using goto statement.

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

```
int main()
```

```
{
```

```
int n;
```

```
printf("Enter any number\n");
```

```
scanf("%d",&n);
```

```
if(n%2==0)
```

```
goto even;
```

```
else
```

```
goto odd;
```

```
even:
```

```
printf("The given number is even");
```

```
goto end;
```

```
odd:
```

```
printf("The given number is odd");
```

```
goto end;
```

```
end:
```

```
}
```

2.break statement:- break is an unconditional statement used to terminate the loops or switch statement.

When it is used in loops(while,do while,for) control comes out of the loop and continues with the next statement in the program.

When it is used in switch statement to terminate the particular case block and then control is transferred to the next statement in the program.

Syntax:-

statement;

break;

Ex: Write a C program based on break statement.

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

```
int main()
{
int i;
for(i=1;i<=5;i++)
{
if(i==4)
break;
printf("%d\t",i);
}
}
```

3.continue statement:- continue is an unconditional statement used to control to be transferred to the beginning of the loop for the next iteration without executing the remaining statements in the program.

Syntax:-

```
statement;
continue;
```

Ex: Write a C program based on continue statement.

```
#include<stdio.h>
int main()
{
int i;
for(i=1;i<=5;i++)
{
if(i==4)
continue;
printf("%d\t",i);
}
}
```

18. Write about command line arguments?

Command line arguments:

The arguments can be passed from operating system command prompt to main(). They are parameters/arguments supplied to the program when it is invoked.

Therefore they are called as Command line arguments.

Command line arguments are two types.

They are

1.argc

2.argv

1.argc-is an argument count that specifies number of arguments in the command line including program name.

2. argv-is an argument vector that specifies array of strings.

Syntax:

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

}
```

19. Write a C program to find the roots of a quadratic equation?

```
#include<stdio.h>
#include<math.h>
int main()
{
float a,b,c,d,root1,root2;
printf(" Enter the values of a,b,c\n");
scanf(" %f %f %f ", &a, &b, &c);
d= (b * b) - (4 * a * c);
if(d >=0)
{
root1=-b+sqrt(d)/(2*a);
root2 =-b-sqrt(d)/(2*a);
printf("root1=%f,root2=%f",root1,root2);
}
else
printf("Roots are imaginary");
}
```

20. Write a C program to find the sum of n natural numbers?

```
#include<stdio.h>
int main()
{
int n,i=1,sum=0;
printf("Enter any number\n");
scanf("%d",&n);
while(i<=n)
{
sum=sum+i;
i++;
}
printf("The sum of n natural numbers is %d",sum);
}
```

21. Write a C program to find the sum of individual digits of a given number?

```
#include<stdio.h>
int main()
{
int digit,n,sum=0;
printf("Enter any number\n");
scanf("%d",&n);
while(n!=0)
{
digit=n%10;
sum=sum+digit;
n=n/10;
}
printf("The sum of individual digits of a given number is %d",sum);
}
```

22. Write a C program to find the reverse of a given number?

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



```
{
int digit,n,reverse=0;
printf("Enter any number\n");
scanf("%d",&n);
while(n!=0)
{
digit=n%10;
reverse=reverse*10+digit;
n=n/10;
}
printf("The reverse of a given number is %d",reverse);
}
```

23. Write a C program to check whether the given number is palindrome or not?

```
#include<stdio.h>
int main()
{
int digit,n,reverse=0,temp;
printf("Enter any number\n");
scanf("%d",&n);
temp=n;
while(n!=0)
{
digit=n%10;
reverse=reverse*10+digit;
n=n/10;
}
if(temp==reverse)
printf("The given number is Palindrome");
else
printf("The given number is not a Palindrome");
}
```

24. Write a C program to check whether the given number is Armstrong or not?

```
#include<stdio.h>
int main()
{
int digit,n,sum=0,temp;
printf("Enter any number\n");
scanf("%d",&n);
temp=n;
while(n!=0)
{
digit=n%10;
sum=sum+(digit*digit*digit);
n=n/10;
}
if(temp==sum)
printf("The given number is Armstrong");
else
printf("The given number is not a Armstrong");
}
```

25. Write a C program to generate first n terms of fibonacci sequence?

```
#include <stdio.h>
int main()
{
    int i,n,first=0,second=1,nextterm;
    printf("Enter the number of terms\n");
    scanf("%d",&n);
    printf("Fibonacci Series is\n ");
    for (i=1;i<=n;i++)
    {
        printf("%d\t",first);
        nextterm=first+second;
        first=second;
        second=nextterm;
    }
}
```

26. Write a C program to generate prime numbers from 1 to n?

```
#include<stdio.h>
int main()
{
    int n,i,j,fact;
    printf("Enter any number\n");
    scanf("%d",&n);
    printf("Prime numbers from 1 to n are\n");
    for(i=1;i<=n;i++)
    {
        fact=0;
        for(j=1;j<=i;j++)
        {
            if(i%j==0)
                fact++;
        }
        if(fact==2)
            printf("%d\t",i);
    }
}
```

27. Write a C program to find the factorial of a given number?

```
#include<stdio.h>
int main()
{
    int n,i,fact=1;
    printf("Enter any number\n");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        fact=fact*i;
    }
    printf("The factorial of a given number is %d",fact);
}
```

28. Write a C program to print the multiplication table of a given number?

```
#include<stdio.h>
int main()
{
    int n,i;
    printf("Enter any number\n");
    scanf("%d",&n);
    printf("Multiplication table of a given number is\n");
    for(i=1;i<=20;i++)
    {
        printf("%d * %d=%d\n",n,i,n*i);
    }
}
```

29. Write a C program to print half pyramid using * ?

```
*
**
***
****
*****
```

```
#include <stdio.h>
int main()
{
    int i, j, rows;
    printf("Enter number of rows: ");
    scanf("%d",&rows);
    for(i=1; i<=rows; i++)
    {
        for(j=1; j<=i; j++)
        {
            printf("* ");
        }
        printf("\n");
    }
}
```

30. Write a C program print half pyramid a using numbers?

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

```
#include <stdio.h>
int main()
{
    int i, j, rows;
    printf("Enter number of rows: ");
    scanf("%d",&rows);
    for(i=1; i<=rows; i++)
    {
        for(j=1; j<=i; j++)
```

```

    {
        printf("%d ",j);
    }
    printf("\n");
}
}

```

UNIT-2

1. Define array and explain how one dimensional array is declared and initialized in C with an example?

1. Array is a collection of elements of the same data type.
2. Arrays can be used to store multiple values.
3. Arrays are the derived data type in C which can store the primary type of data such as int, char, double, float, etc.
4. Array range starts from 0 to n-1.
5. Array elements are stored in contiguous memory locations or consecutive memory locations or successive memory locations.

1.One dimensional arrays (1DA) :-

If an array contains one subscript then it is called one dimensional array.

Declaration of one dimensional arrays (1DA)

Syntax:

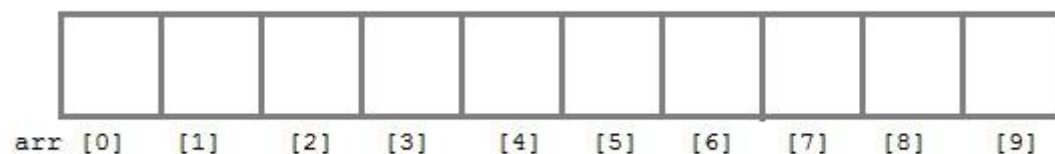
datatype arrayname[size];

In the above syntax data type maybe any basic data type such as int,float, etc.

arrayname is any valid identifier.

size indicates number of elements can be stored in an array name.

Example: int arr[10];



Here int is the data type, arr is the name of the array and 10 is the size of array. It means array arr can only contain 10 elements of int type. Index of an array starts from 0 to size-1 i.e first element of arr array will be stored at arr[0] address and last element will occupy arr[9].

Initialization (or) Assigning values to one dimensional arrays (1DA) :

Syntax :

datatype arrayname[size] = { value1, value2,....., value n};

Example:

int x[5]={10, 20, 30, 40, 50};

In the above example five integer elements can be stored in the array name 'x'

Here x[0] refers to the 1st element stored in the array i.e.10

x[1] refers to the 2nd element stored in the array i.e.20

x[2] refers to the 3rd element stored in the array i.e.30

x[3] refers to the 4th element stored in the array i.e.40

x[4] refers to the 5th element stored in the array i.e.50

Example: Program to Reading and Storing 5 elements in one dimensional array.

```
#include<stdio.h>
main()
{
    int a[10],i;
    printf("Enter 10 integers\n");
    for(i=0;i<10;i++)
    {
        scanf("%d",&a[i]);
    }
    printf("Displaying integers\n");
    for(i=0;i<10;i++)
    {
        printf("%d\n",a[i]);
    }
}
```

2. Write a C program to find both the largest and smallest elements from the given array?

```
#include<stdio.h>
int main()
{
    int x[5],i,large,small;
    printf(" Enter any five integer array elements\n");
    for(i=0;i<5;i++)
    {
        scanf("%d",&x[i]);
    }
    large=x[0];
    small=x[0];
    for(i=0;i<5;i++)
    {
        if(x[i]>large)
            large=x[i];
        if(x[i]<small)
            small=x[i];
    }
    printf("The largest element from the given array is %d\nThe smallest element from the given array is %d",large,small);
}
```

3. Write a C program to sort the array elements in ascending order?

```
#include <stdio.h>
int main()
{
    int A[50],i,j,temp,n;
    printf("Enter the size of an array\n");
    scanf("%d",&n);
```

```

printf("Enter the elements of an Array\n");
for(i=0;i<n;i++)
{
    scanf("%d",&A[i]);
}
for(i=0;i<n;i++)
{
    for(j=i+1;j<n;j++)
    {
        if(A[i]>A[j])
        {
            temp = A[i];
            A[i] = A[j];
            A[j] = temp;
        }
    }
}
printf("Array elements in Ascending Order are\n");
for(i=0;i<n;i++)
{
    printf("%d",A[i]);
}
}

```

4. Explain how two dimensional arrays are declared and initialized in C with an example?

Two Dimensional Arrays (2DA):

If an array contains two subscripts then it is called two dimensional arrays.

It is also called multi dimensional arrays.

Two dimensional array elements are stored in consecutive memory locations.

Two dimensional arrays are mainly used for performing matrix operations like matrix addition, subtraction, multiplication, transpose, etc.

Declaration of Two Dimensional Arrays (2DA):

datatype arrayname[size1][size2] ;

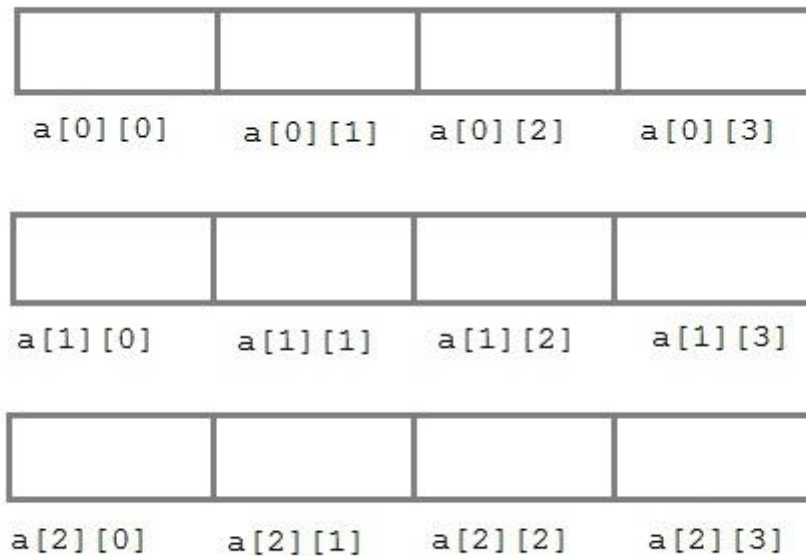
In the above syntax data type maybe any basic data type such as int, float, etc.

arrayname is any valid identifier.

size1 indicates no. of rows and size2 indicates no. of columns stored in an arrayname.

Example:

```
int a [3][4];
```



Initialization(or)Assigning values to two dimensional arrays (2DA) :

Syntax :

`datatype arrayname[size1][size2] = { value1, value2,....., value n};`

Example:

`int x[2][2] = {10,20,30,40};`

In the above example 2 rows and 2 columns can be stored in the array name 'x'

Here `x[0][0]` refers to the 1st element stored in the array i.e.10

`x[0][1]` refers to the 2nd element stored in the array i.e.20

`x[1][0]` refers to the 3rd element stored in the array i.e.30

`x[1][1]` refers to the 4th element stored in the array i.e.40

Example: Program to Reading and Storing elements in a matrix and printing it.

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

```
main()
```

```
{
```

```
int a[3][3],i,j;
```

```
printf("Enter the elements of matrix a\n");
```

```
for(i=0;i<3;i++)
```

```
{
```

```
for(j=0;j<3;j++)
```

```
{
```

```
scanf("%d",&a[i][j]);
```

```
}
```

```
}
```

```
printf("Printing the elements\n");
```

```
for(i=0;i<3;i++)
```

```
{
```

```
for(j=0;j<3;j++)
```

```
{
```

```
printf("%d\t",a[i][j]);
```

```
}
```

```
printf("\n");
```

```
}
```

```
}
```

5. Write a C program to perform transpose of a matrix of order 3*3?

```
#include<stdio.h>
int main()
{
int A[3][3],Transpose[3][3],i,j;
printf("Enter the values of elements of matrix A\n");
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
scanf("%d",&A[i][j]);
}
}
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
Transpose[i][j]=A[j][i];
}
}
printf("The Transpose of matrix A is\n");
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
printf("%d\t",Transpose[i][j]);
}
printf("\n");
}
}
```

6. Write a C program to perform addition of two matrices and print the result in another matrix of order 3*3?

```
#include<stdio.h>
int main()
{
int A[3][3], B[3][3], C[3][3],i,j;
printf("Enter the elements of matrix A\n");
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
scanf("%d",&A[i][j]);
}
}
printf("Enter the elements of matrix B\n");
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
scanf("%d",&B[i][j]);
}
}
}
```



```
for(i=0;i <3;i++)
{
for(j= 0;j <3;j++)
{
C[i][j]=A[i][j]+B[i][j];
}
}
printf("The result in C matrix is \n");
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
printf("%d\t",C[i][j]);
}
printf("\n");
}
}
```

7. Write a C program to perform multiplication of two matrices and print the result in another matrix of order 3*3?

```
#include<stdio.h>
int main()
{
int A[3][3], B[3][3], C[3][3],i,j,k;
printf("Enter the elements of matrix A\n");
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
scanf("%d",&A[i][j]);
}
}
printf("Enter the elements of matrix B\n");
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
scanf("%d",&B[i][j]);
}
}
for(i =0;i <3;i++)
{
for(j= 0;j <3;j++)
{
C[i][j]=0;
for(k= 0;k<3; k++ )
{
C[i][j]=C[i][j]+(A[i][k]*B[k][j]);
}
}
}
printf("The result in C matrix is \n");
for(i=0;i<3;i++)
```

```

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

```

8. Define string and explain how strings are declared and initialized in C?

- 1.String is a group of characters enclosed within double quotation marks
- 2.String is also called array of characters or character arrays.
- 3.Character Array elements are stored in contiguous memory locations or consecutive memory locations or successive memory locations.
- 4.The string should end with '\0' (null character) in C language.
- 5.The size of the string is equal to no.of characters in the string + 1 (required for null character('\0')).

Declaration of strings/character arrays:

Syntax:

```
datatype arrayname[size];
```

In the above syntax data type must be 'char' data type.

array name is any valid identifier

size indicates no.of characters in the string + 1 (required for null character('\0')).

Example: char name[20];

Initialization of strings/character arrays:

Syntax: datatype arrayname[size]={character1,character2,...,character n};

Example:

```
char city[6]="DELHI";
```

(or)

```
char city[6]={'D','E','L','H','I','\0'};
```

city[0] refers to 1st character in string i.e. D

city[1] refers to 2nd character in string i.e. E

city[2] refers to 3rd character in string i.e. L

city[3] refers to 4th character in string i.e. H

city[4] refers to 5th character in string i.e. I

city[5] refers to 6th character in string i.e. '\0'.

Example:

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

```
int main()
```

```

{
    char name[50];
    printf("Enter your name\n ");
    gets(name);
    printf("Your name is\n ");
    puts(name);
}

```

9. Explain any 5 string handling/manipulation functions in C with an example programs?

String Manipulation Functions/String Handling Functions:-

All the string handling functions are pre-defined in the system library i.e., in the header file `#include<string.h>`

The following are commonly used string handling functions in C are

1. `strcpy()`
2. `strcat()`
3. `strrev()`
4. `strlen()`
5. `strlwr()`
6. `strupr()`
7. `strcmp()`

1.strcpy():-

This function is used to copy one string into another string.

Syntax: `strcpy(string1,string 2);`

Here, string 2 is copied into string 1 and after copying both the contents of string 1 and string 2 are same.

Example: C program to copy one string into another string using `strcpy()`.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
    char name1[20],name2[20];
    printf("Enter the first name\n");
    gets(name1);
    printf("Enter the second name\n");
    gets(name2);
    strcpy(name1,name2);
    printf("After copying the name1=%s,name2=%s",name1,name2);
}
```

2.strcat():-

This function is used to combine two strings.

Syntax: `strcat (string1,string2);`

Here, string2 is added to the end of the string1.

Example : C program to combine two strings using `strcat()`.

```
#include<stdio.h>
#include<string.h>
int main()
{
    char name1[20],name2[20];
    printf("Enter the first name\n");
    gets(name1);
    printf("Enter the second name\n");
    gets(name2);
    strcat(name1,name2);
    printf("After concatenation the name is %s",name1);
}
```

3.strrev() :-

This function is used to reverse a given string.

Syntax: `strrev(string);`

Example: C program to find the reverse of given string using `strrev()`.

```
#include<stdio.h>
#include<string.h>
int main()
{
char name[20];
printf("Enter any name\n");
gets(name);
strrev(name);
printf("The reverse of a given name is %s",name);
}
```

4.strlen():-

This function is used to find the length of the given string.

This function does not include the '\0'(null character).

Syntax: `strlen(string);`

Example: C program to find the length of a given string using `strlen()`.

```
#include<stdio.h>
#include<string.h>
int main()
{
char name[20];
int x;
printf("Enter any name\n");
gets(name);
x=strlen(name);
printf("The length of a given string is %d",x);
}
```

5.strlwr():-

This function is used to convert any uppercase letters into lowercase letters.

Syntax: `strlwr(string);`

Example: C program to convert any uppercase letters into lowercase letters using `strlwr()`.

```
#include<stdio.h>
#include<string.h>
int main()
{
char name [20];
printf("Enter any name\n");
gets(name);
strlwr(name);
printf("The converted name is %s",name);
}
```

6.strupr():-

This function is used to convert any lowercase letters into uppercase letters.

Syntax: `strupr(string);`

Example: C program to convert any lowercase letters into letters uppercase using `strupr()`.

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

```
int main()
{
char name [20];
printf("Enter any name\n");
gets(name);
strupr(name);
printf("The converted name is %s",name);
}
```

7.strcmp():-

This function is used to compare two strings character by character based on ASCII values (American Standard Code for Information Interchange) and returns zero when both the strings are equal.

Syntax: strcmp(string1,string2);

Example : C program to check whether the two strings are equal or not.

```
#include<stdio.h>
#include<string.h>
int main()
{
char name1[20],name2[20];
printf("Enter the first name\n");
gets(name1);
printf("Enter the second name\n");
gets(name2);
if(strcmp(name1,name2)==0)
printf("Both the given strings are equal");
else
printf("Both the given strings are not equal");
}
```

10. Write a C program to check whether the given string is palindrome or not?

```
#include<stdio.h>
#include<string.h>
int main()
{
char string[20];
int i,length,flag=1;
printf("Enter any string\n");
scanf("%s",string);
length=strlen(string);
for(i=0;i<length;i++)
{
if(string[i]!=string[length-i-1])
{
flag=0;
break;
}
}
if(flag==1)
printf("The given string is a palindrome");
else
```

```
printf("The given string is not a palindrome");
}
```

11. Explain array of strings/two dimensional strings with an example program?

Two dimensional strings/Array of strings/ Two dimensional character arrays

Using one dimensional string only a single string was accepted and processed.

Sometimes, it becomes necessary to process a group of strings. In such situations we need a two dimensional character arrays.

Declaration of Two dimensional strings:

Syntax:

```
datatype arrayname[size1][size2];
```

In the above syntax data type must be 'char' data type.

array name is any valid identifier.

size1 indicates no. of strings and size2 indicates no. of characters of each string.

Example: char name[5][20];

Initialization of Two dimensional strings:

Syntax: datatype arrayname[size1][size2]={string1,string2,...,string n};

Example:

```
char name[5][20]={"dileep","sravani","mehta","likith","siddarath"};
```

name[0] refers to 1st string i.e.dileep

name[1] refers to 2nd string i.e.sravani

name[2] refers to 3rd string i.e.mehta

name[3] refers to 4th string i.e.likith

name[4] refers to 5th string i.e.siddarath

Example: Write a c program to read and print any 5 names on the monitor.

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

```
int main()
```

```
{
```

```
char name[5][20];
```

```
int i;
```

```
printf("Enter any 5 names");
```

```
for(i=0;i<5;i++)
```

```
{
```

```
scanf("%s",&name[i]);
```

```
}
```

```
printf("Entered 5 names are\n");
```

```
for(i=0;i<5;i++)
```

```
{
```

```
printf("%s\n",name[i]);
```

```
}
```

```
}
```

12. Write a C program to sort the given strings in alphabetical order?

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

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

```
int main()
```

```
{
```

```
char name[10][20],temp[20];
```

```
int n,i,j;
```

```
printf("Enter the number of strings\n");
```

```

scanf("%d",&n);
printf("Enter the strings\n");
for(i=0;i<n;i++)
{
scanf("%s",name[i]);
}
for(i=0;i<n-1;i++)
{
for(j=0;j<n-1;j++)
{
if(strcmp(name[j],name[j+1])>0)
{
strcpy(temp,name[j]);
strcpy(name[j],name[j+1]);
strcpy(name[j+1],temp);
}
}
}
printf("The sorted order of strings are\n") ;
for(i=0;i<n;i++)
{
printf("%s\n",name[i]);
}
}

```

13. Define a pointer and explain how the pointer variable is declared and initialized in C?

Pointer is a variable that stores the address of another variable.

Pointer is used to allocate the memory dynamically at the run time.

Declaration of a pointer:

datatype *variable;

Example1: int *p;

Here p is a pointer variable which always points to integer data type.

Example2: float *p;

Here p is a pointer variable which always points to real or floating point data type.

Example3: char *p;

Here p is a pointer variable which always points to character data type.

Initialization of a pointer:

The way of allocating the address of a variable to a pointer variable is known as pointer initialization.

Pointer can be initialized during the declaration as follows

Syntax: pointerVariableName = &variableName ;

Example:

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

Reference operator (&)

& is called reference operator/address operator which specifies where the value would be stored.

Dereference operator/Indirection operator (*)

* is called indirection operator or dereferencing operator which indicates value at address.

Example: C program to print the values and its corresponding addresses using pointers.

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

```
main()
```

```
{
```

```
int *p,x=10;
```

```
p=&x;
```

```
printf("the address of p is %u\n",&p);
```

```
printf("the address of x is %u\n",&x);
```

```
printf("the value of p is %u\n",p);
```

```
printf("the value of x is %d\n",x);
```

```
printf("the value at p is %d\n",*p);
```

```
printf("the value at x is %d\n",*(&x));
```

```
}
```

14. Write short notes on the following

1. Pointer Arithmetic

2. Pointers to pointers

3. void pointers

1.Pointer Arithmetic/Address Arithmetic:

When we increment/decrement a pointer variable its value is incremented or decremented by the length the data type that points to. This length is called scale factor.

Example: Write a c program to perform pointer arithmetic or increment a pointer.

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

```
int main()
```

```
{
```

```
int x,*p;
```

```
float y,*q;
```

```
char z,*r;
```

```
p=&x;
```

```
q=&y;
```

```
r=&z;
```

```
printf("the value of p is %u\n",p);
```

```
printf("the value of q is %u\n",q);
```

```
printf("the value of r is %u\n",r);
```

```
p++;
```

```
q++;
```

```
r++;
```



```
printf("After incrementation the value of p is %u\n",p);
printf("After incrementation the value of q is %u\n",q);
printf("After incrementation the value of r is %u",r);
}
```

2. Pointers to pointers/double pointer:

Pointer variable stores the address of another pointer variable is called double pointer or pointers to pointers.

Declaration of pointers to pointers/double pointer:

syntax:

datatype **variable;

Example: int **p;

Example: Write a c program to print the values and corresponding addresses on the monitor using pointers to pointers or double pointers.

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

```
int main()
```

```
{
```

```
int x=20,*p,**q;
```

```
p=&x;
```

```
q=&p;
```

```
printf("the address of x is %u\n",&x);
```

```
printf("the address of p is %u\n",&p);
```

```
printf("the address of q is %u\n",&q);
```

```
printf("the value of x is %d\n",x);
```

```
printf("the value of p is %u\n",p);
```

```
printf("the value of q is %u\n",q);
```

```
printf("the value at x is %d\n",*(&x));
```

```
printf("the value of p is %d\n",*p);
```

```
printf("the value of q is %d",**q);
```

```
}
```

3. void pointers

In C programming language, a void pointer is a a generic pointer variable used to store the address of a variable of any data type. That means single void pointer can be used to store the address of integer variable, float variable, character variable.

Suppose we have to declare integer pointer, character pointer and float pointer then we need to declare 3 pointer variables. But using void pointer we can declare single pointer variable which can act as integer pointer, character pointer and float pointer.

Declaration of void pointer

syntax:

void *variable;

Example program:

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

```
int main()
```

```
{
```

```
void *p;
```

```
int i;
```

```
char j;
```

```
float k;
```

```
printf("the address of p is %u\n",&p);
```

```

printf("the address of i is %u\n",&i);
printf("the address of j is %u\n",&j);
printf("the address of k is %u\n",&k);
p=&i;
printf("the value of p is %u\n",p);
p=&j;
printf("the value of p is %u\n",p);
p=&k;
printf("the value of p is %u\n",p);
}

```

15. Explain pointers with arrays with an example program?

Pointers and Arrays:

When an array is declared, compiler allocates sufficient amount of memory to contain all the elements of the array. The base address is the location of the first element (index 0) of the array.

Declaration of pointers and arrays:

Syntax: datatype arrayname[size],*variable;

Example: Suppose we declare an array arr,

```
int arr[5]={ 1, 2, 3, 4, 5 };
```

Assuming that the base address of arr is 1000 and each integer requires two bytes, the five elements will be stored as follows

element	arr[0]	arr[1]	arr[2]	arr[3]	arr[4]
Address	1000	1002	1004	1006	1008

Here variable arr will give the base address, which is a constant pointer pointing to the element, arr[0]. Therefore arr is containing the address of arr[0] i.e 1000.

We can declare a pointer of type int to point to the array arr.

```
int *p;
p = &arr[0];
```

Now we can access every element of array arr using p++ to move from one element to another.

Example: Write a c program to read and print any 5 integer array elements using pointers.

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

```

{
int x[5],*p,i;
p=&x[0];
printf(" Enter any five integer array elements\n");
for(i=0;i<5;i++)
{
scanf("%d",&x[i]);
}
printf(" The 5 integer array elements are \n");
for(i=0;i<5;i++)
{
printf("%d is stored at address %u\n",*p,p);
p++;
}
}

```

16. Explain pointers with strings with an example program?

Pointers and Strings:

When a character array is declared, compiler allocates sufficient amount of memory to contain all the characters of an array. The base address is the location of the first element (index 0) of the character array.

Declaration of pointers and strings:

Syntax: datatype arrayname[size],*variable;

Here data type must be char data type.

Example: Write a C program to read and print any string using pointers.

```

#include<stdio.h>
#include<conio.h>
int main()
{
char name[20],*p;
int i;
p=&name[0];
printf(" Enter any string\n");
scanf("%s",&name);
for(i=0;name[i]!='\0';i++)
{
printf("%c is stored at address %u\n",*p,p);
p++;
}
}

```

17. Write a C program to find the sum of all elements stored in array using pointers?

```

#include<stdio.h>
int main()
{
int x[5],*p,i,sum=0;
p=&x[0];
printf(" Enter any 5 integer array elements\n");

```

```
for(i=0;i<5;i++)
{
scanf("%d",&x[i]);
}
for(i=0;i<5;i++)
{
sum=sum+*p;
p++;
}
printf(" The sum of array elements using pointers is %d",sum);
}
```

18. Write a C program to find the length of the given string using pointers?

```
#include<stdio.h>
int main()
{
char string[20],*p;
int i,length=0;
clrscr();
printf("Enter any string\n");
scanf("%s",&string);
p=&string[0];
while(*p!='\0')
{
p++;
length++;
}
printf("The length of the given string is %d",length);
}
```

UNIT-3

1. Define a function and explain the different types of functions with an example program?

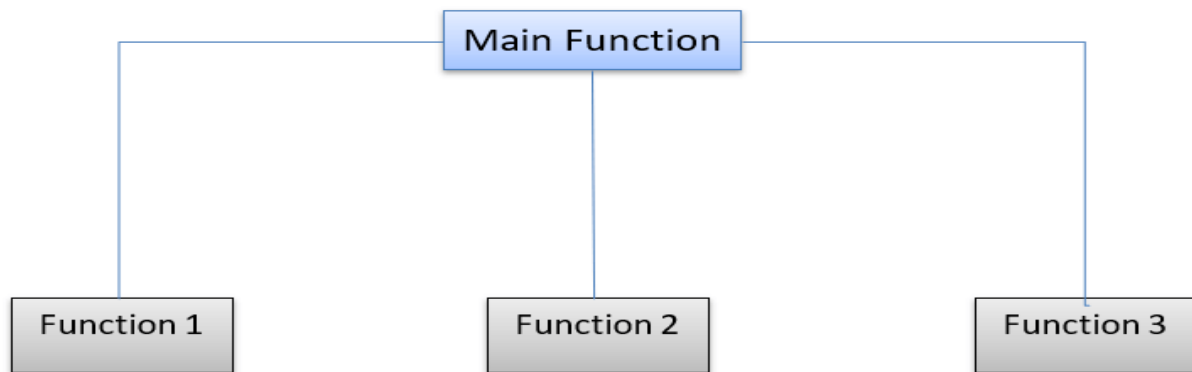
A Function is a sub program/self-contained block of one or more statements that performs a special task when called.

Every C program execution starts with main().

main() calls other functions to share the work.

The large programs can be divided into small programs using functions. Hence, a large project can be divided among many programmers.

Data reusability is the main achievement of C functions.



How function works: Whenever a function is called, control passes to the called function and working of the calling function is temporarily stopped. When the execution of the called function is completed, then control return back to the calling function and executes the next statements in a program.

Types of functions:

There are two types:

- 1) Library functions/Standard functions
- 2) User defined functions

1. Library functions: Library functions are fixed and pre defined meaning in C language.

Ex: printf(),scanf(),sqrt(),pow() etc.

2. User defined functions: These functions are developed by the user at the time of writing a program.

Eg: main(),MRCET(),add(),factorial() etc.

Every function in C has the following 3 elements

1. Function Declaration (Function Prototype)
2. Function Call
3. Function Definition

1. Function Declaration (Function Prototype):

The function declaration tells the compiler about function name, the data type of the return value and parameters. The function declaration is also called a function prototype.

Syntax:

returntype functionname(parameters list);

2. Function Call:

The function call tells the compiler when to execute the function definition. When a function call is executed, the execution control jumps to the function definition where the actual code gets executed and returns to the same functions call once the execution completes.

Syntax:

functionname(parameters list);

3. Function Definition:

The function definition provides the actual code of that function. The function definition is also known as the body of the function. The actual task of the function is implemented in the function definition. The function definition is performed before the main function or after the main function.

Syntax:

```

returntype    functionname(datatype    argument1,datatype    argument2,...datatype
argument n )
{
body of a function;
return(expression);
}

```

In the above syntax returntype may be any basic data type such as int, float, char etc. return statement is used to send a value back to the calling function.

Example:

```

main() //calling function
{
    MRCET(A,B,C); //called function
}
MRCET(X,Y,Z)
{
body of a function;
}

```

Actual arguments/Actual parameters: The arguments of the calling function are called actual arguments. In the above example A,B,C are actual arguments.

Formal arguments/Formal parameters: The arguments of the called function are called formal arguments. In the above example X,Y,Z are called formal arguments.

Example: Write a C program to find the addition of two numbers using functions.

```

#include<stdio.h>
int add(int x, int y);
int main()
{
    int a,b,c;
    printf("Enter the values of a,b \n");
    scanf(" %d%d",&a,&b);
    c=add(a,b);
    printf("The sum of a,b is %d",c);
}
int add(int x, int y)
{
    return (x+y);
}

```

2. Explain the categories of functions based on the arguments and return values?

Function Categories (Based on Arguments and Return Values)

There are 4 types

1. Functions with arguments and functions with return values.
2. Functions without arguments and functions with no return values.
3. Functions with no arguments and functions with return value.
4. Functions with arguments and functions with no return values.

1. Functions with arguments and functions with return values

Functions with arguments i.e. the called function receive the data from the calling function.

Function with return values i.e. calling function receives the value from the called function.

In effect there is a data transfer between calling and called function.

Example program

```
#include<stdio.h>
int add(int x, int y);
int main()
{
    int a,b,c;
    printf("enter the values of a,b\n");
    scanf("%d%d",&a&&b);
    c=add(a,b);
    printf("the sum is %d",c);
}
int add(int x , int y)
{
    return(x+y);
}
```

2. Functions with no arguments and functions with no return values.

Functions with no arguments i.e. the called function does not receive any data from calling function.

Functions with no return values i.e. the calling function does not receive any value from the called function.

In effect there no data transfer between the calling and called function.

Example program

```
#include<stdio.h>
void add();
int main()
{
    add();
}
void add()
{
    int a,b,c;
    printf("Enter the values of a,b\n");
    scanf("%d%d",&a&&b);
    printf("the sum is %d",a+b);
}
```

3. Functions with no arguments and functions with return values

Functions with no arguments i.e. the called function does not receive any data from the calling function.

Functions with return values i.e. the calling function receives the value from the called function.

In effect there is no data transfer from calling function to called function, but there is a data transfer from the called function to calling function.

Example program

```
#include<stdio.h>
int add();
int main()
{
    int c;
    c=add();
}
```

```
printf("the sum is %d",c);
}
int add()
{
int a,b;
printf("Enter the values of a,b\n");
scanf("%d%d",&a,&b);
return(a+b);
}
```

4.Functions with arguments and functions with no return values

Functions with arguments i.e. the called function receives the data from the calling function.

Functions with no return values i.e. the calling function does not receives the value from the called function.

In effect there is a data transfer from calling function to called function, but there is no data transfer from the called function to calling function.

Example program

```
#include<stdio.h>
void add(int x, int y);
int main()
{
int a,b;
printf("Enter the values of a,b\n");
scanf("%d%d",&a,&b);
add(a,b);
}
void add(int x,int y)
{
printf("The sum is %d",x+y);
}
```

3. Explain parameter passing techniques with an example programs?

Parameter passing:

The process of information interchanging between the calling function and the called function is known as parameter parsing.

There are two methods

- 1) Call by value/Pass by value
- 2) Call by address/Call by reference/Pass by address

1.Call by value:

Sending or passing the values as actual arguments to the called function is known as call by value.

In this method the changes are made in formal arguments of the called function does not effect the actual arguments of the calling function.

Ex: Write a C program to interchange the values of two variables using call by value.

```
#include<stdio.h>
int swap(int x, int y);
int main()
{
int a=10,b=20;
printf("Before swapping a=%d, b=%d \n", a,b);
```



```

swap(a,b);
printf(" After swapping a=%d, b=%d", a,b);
}
int swap(int x, int y)
{
int temp;
printf(" Before swapping x=%d, y=%d \n",x,y);
temp=x;
x=y;
y=temp;
printf(" After swapping x=%d, y=%d \n",x,y);
}

```

2.Call by address:

Sending or passing addresses as an actual arguments to the called function is known as call by address.

In this method changes are made in formal arguments of the called function effects the actual arguments of the calling function.

Ex: Write a C program to interchange the value of two variable using call by address.

```

#include<stdio.h>
int swap(int *x, int *y);
int main()
{
int a =10,b=20;
printf("Before swapping a=%d, b=%d \n",a,b);
swap(&a,&b);
printf(" After swapping a=%d, b=%d",a,b);
}
int swap(int *x, int *y)
{
int temp;
temp = *x;
*x = *y;
*y = temp;
}

```

4. Define scope and differentiate local and global variables in C?

Variables can be accessed based on their scope. The scope of a variable decides the portion of a program in which the variable can be accessed. The variable scope defines the visibility of variable in the program. Scope of a variable depends on the position of variable declaration.

Local variables:

- 1.The variables which are declared within the body of a function are called local variables.
- 2.Local variables are also called internal variables.
- 3.Local variables are cannot be accessed by other functions in a program.
- 4.The default value of a local variable is garbage value.
- 5.Local variables are used to store the data values temporarily.
- 6.The life time of a local variable is till the control remains within the function.

Global variables:

- 1.The variables which are declared outside the main() function are called global variables.
- 2.Global variables are also called external variables.
- 3.Global variables are accessed by all the functions in the program.
- 4.The default value of a global variable is zero.
- 5.Global variables are used to store the data values permanently.
- 6.The life time of a global variable is till the program terminates.

Ex: Write a C program to give an example based on local and global variables.

```
#include<stdio.h>
int x=50,y =100;
int main()
{
int a =10,b=20;
printf("Local variables are a=%d,b=%d \n", a,b);
printf(" Global variables are x=%d,y=%d\n",x,y);
MRCET();
}
MRCET()
{
int a,b;
printf("Local variables are a=%d,b=%d \n",a,b);
printf("Global variables are x=%d,y=%d \n",x,y);
}
```

5. Explain the different types of storage classes available in C?

Storage Classes:

Storage class of a variable defines the following terms:

1. Where the variable would be stored.
2. What is the scope of the variable.
3. What is the default value of the variable.
4. What is the life time of the variable.

Based on the above terms storage classes are of four types:

1. Automatic storage class
2. Register storage class
3. Static storage class
4. External storage class

1.Automatic storage class:

1. It is indicated by the keyword 'auto'.
2. It is stored in main memory.
3. The scope of auto storage class is local.
4. The default value of auto storage class is garbage value.
5. Lifetime of auto storage class is till the control remains within the function.
6. Syntax: auto datatype variable;
7. Example: auto int a;

2.Register storage class:

1. It is indicated by the keyword 'register'.
2. It is stored in CPU registers.
3. The scope of register storage class is local.
4. The default value of register storage class is garbage value.
5. Lifetime of register storage class is till the control remains within the function.

6. Syntax: register datatype variable;
7. Example: register int a;

3.Static storage class:

1. It is indicated by the keyword 'static'.
2. It is stored in main memory.
3. The scope of static storage class is local.
4. The default value of static storage class is garbage value.
5. Lifetime of static storage class is till the control remains within the function.
6. Syntax: static datatype variable;
7. Example: static int a;

4.External storage class:

1. It is indicated by the keyword 'extern'.
2. It is stored in main memory.
3. The scope of extern storage class is global.
4. The default value of extern storage class is zero.
5. Lifetime of extern storage class is till the program terminates.
6. Syntax: extern datatype variable;
7. Example: extern int a;

6. Define recursion and write a C program to find factorial of a given number using recursion?

Recursion:

When a called function in turn calls the same function then chaining occurs. Recursion is a special case of this process or a repetitive process where a function calls itself.

Example: C program to find the factorial of a given number using recursion.

```
#include<stdio.h>
int factorial(int x);
int main()
{
    int r,n;
    printf("Enter any number\n");
    scanf("%d",&n);
    r=factorial(n);
    printf("the factorial of the given number is %d",r);
}
int factorial(int x)
{
    int fact;
    if(x==0)
        return(1);
    else
        fact=x*factorial(x-1);
    return(fact);
}
```

7. Write a C program to find GCD of a given number using recursion?

```
#include<stdio.h>
int gcd(int x,int y);
int main()
{
    int a,b,r;
    printf("Enter the two numbers\n");
    scanf("%d%d",&a,&b);
```

```

r=gcd(a,b);
printf("The GCD of a,b is %d",r);
}
int GCD(int x,int y)
{
if(y==0)
return(x);
else
return GCD(y,x%y);
}

```

8. Explain passing arrays as parameters to functions in C with an example program?

In C programming, you can pass an entire array as parameter to functions. To pass an entire array to a function, only the name of the array is passed as an argument.

Syntax:

```

returntype functionname(datatype arrayname[])
{
Body of a function;
}

```

Example:

```

#include<stdio.h>
int display(int x[]);
main()
{
int a[5],i;
printf("Enter 5 integers\n");
for(i=0;i<5;i++)
{
scanf("%d",&a[i]);
}
display(a);
}
int display(int x[])
{
int i;
printf("Displaying integers\n");
for(i=0;i<5;i++)
{
printf("%d\n",x[i]);
}
}

```

9. Explain pointers to functions with an example program?

Pointers to Functions:

Pointers can be passed as arguments to a called function.

Passing pointers to a function is called call by reference/call by address.

Pointer arguments are useful in functions because they allow to access the original data in the calling function.

Example: Write a C program to interchange the values of two variables using call by address.

```
#include<stdio.h>
int swap(int *x, int *y);
int main()
{
    int a =10,b=20;
    printf("Before swapping a=%d, b=%d \n",a,b);
    swap(&a,&b);
    printf(" After swapping a=%d, b=%d",a,b);
}
int swap(int *x, int *y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
```

10. Explain briefly about dynamic memory management functions?

Dynamic Memory Allocation

The process of allocating memory during program execution is called dynamic memory allocation. All the memory management functions are found in standard library file (stdlib.h).

Dynamic Memory Allocation Functions in C

C language offers 4 dynamic memory allocation functions. They are,

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

1.malloc() function

1.malloc() function is used to allocate space in memory during the execution of the program.

2.malloc() does not initialize the memory allocated during execution. It carries garbage value.

3.malloc() function returns null pointer if it couldn't able to allocate requested amount of memory.

Syntax

malloc ()	malloc (number *sizeof(int));
------------	-------------------------------

2.calloc() function

calloc () function is also like malloc () function. But calloc () initializes the allocated memory to zero. But, malloc() doesn't.

Syntax

calloc ()

calloc (number, sizeof(int));

3. realloc () function

1. realloc () function modifies the allocated memory size by malloc () and calloc () functions to new size.

2. If enough space doesn't exist in memory of current block to extend, new block is allocated for the full size of reallocation, then copies the existing data to new block and then frees the old block.

Syntax

realloc ()

realloc (pointer_name, number * sizeof(int));

4. free () function

free () function frees the allocated memory by malloc (), calloc (), realloc () functions and returns the memory to the system.

Syntax

free ()

free (pointer_name);

UNIT-4**1. Define a structure and explain about declaration, initialization and accessing of structure members in C with an example program?**

Structure is a collection of elements of different data types.

Declaration of a Structure:-Syntax:

struct tagname

{

datatype member1;

datatype member2;

datatype member3;

.

.

datatype member n;

};

In the above syntax struct is a keyword that declares/stores the members or fields of a structure.

tag name is name of the structure.

datatype maybe any basic datatype such as int, float, char etc.

member1, member2,, member n are structure members or fields of a structure.

The body of a structure should be end with semicolon(;).

Example:

struct student

{

int rno;

```
char name[20];
char branch[20];
float marks;
};
```

Declaration of a structure variable:-

Syntax:-

```
struct tagname variable;
```

Example:

```
struct student s;
```

Initialization of a structure variable:-

Syntax:-

```
struct tagname variable={member1,member2,...member n};
```

Example:

```
struct student s={501,"Dinesh","cse",100.5};
```

Member operator or dot operator(.):-

To access the each member of a structure using the structure variable we have to use dot operator (.) in c language.

Syntax:-

```
Structure variable.Structure member
```

Example Program:

Example: Define a structure type student that would contain roll number,name,branch and marks. Write a C program to read this information from the keyboard and print the same on the monitor.

```
#include<stdio.h>
struct student
{
int rno;
char name[20];
char branch[20];
float marks;
};
int main()
{
struct student s;
printf("Enter the student rno,name,branch and marks\n");
scanf("%d%s%s%f",&s.rno,s.name,s.branch,&s.marks);
printf("the student details are\n");
printf("%d\t%s\t%s\t%f",s.rno,s.name,s.branch,s.marks);
}
```

2. Write a C program to calculate total and percentage of a student using structure?

```
#include<stdio.h>
struct student
{
int rno;
char name[20];
int m1,m2,m3,m4,m5,total,maxmarks,percentage;
};
int main()
{
```

```

struct student s;
printf("Enter the student rno,name,m1,m2,m3,m4,m5 and max marks\n");
scanf("%d%s%d%d%d%d%d", &s.rno, s.name, &s.m1, &s.m2, &s.m3, &s.m4, &s.m5, &s.maxmarks);
s.total=s.m1+s.m2+s.m3+s.m4+s.m5;
s.percentage=(s.total/s.maxmarks)*100;
printf("rno=%d\nname=%s\nm1=%d\nm2=%d\nm3=%d\nm4=%d\nm5=%d\ntotal=%d\npercentage=%d", s.rno, s.name, s.m1, s.m2, s.m3, s.m4, s.m5, s.total, s.percentage);
}

```

3. Explain array of structures with an example program?

Array of Structures:

The same structure is applied to a group of people or group of items then array of structures is used.

Syntax:

```
struct tagname arrayname[size];
```

Example: Define a structure type student that would contain roll number, name, branch and marks. Write a C program to read this information for 5 students from keyboard and print the same on monitor.

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

```
struct student
```

```
{
```

```
int rno;
```

```
char name[20];
```

```
char branch[20];
```

```
float marks;
```

```
};
```

```
int main()
```

```
{
```

```
struct student s[5];
```

```
int i;
```

```
printf("Enter any 5 student details \n");
```

```
for(i=0;i<5;i++)
```

```
{
```

```
printf("Enter the student rno,name,branch,marks\n");
```

```
scanf("%d%s%s%f", &s[i].rno, s[i].name, s[i].branch, &s[i].marks);
```

```
}
```

```
printf("the 5 students details are\n");
```

```
for(i=0;i<5;i++)
```

```
{
```

```
printf("%d\t%s\t%s\t%f\n", s[i].rno, s[i].name, s[i].branch, s[i].marks);
```

```
}
```

4. Explain structures and pointers with an example program?

Pointers to Structures:

A pointer variable which points to a structure is called structure pointers/pointers to structures.

Syntax:

```
struct tagname *variable;
```

Selection operator/arrow operator (->)

To access the each member of a structure using pointer variable we have use arrow operator (->) in c language.

Syntax:

Pointer variable ->structure member

Example: Define a structure type student that would contain roll number,name,branch and marks. Write a c program to read this information from keyboard and print the same on monitor using structure pointer or (->)operator.

```
#include<stdio.h>
struct student
{
int rno;
char name[20];
char branch[20];
float marks;
};
int main()
{
struct student s,*p;
p=&s;
printf("Enter the student rno,name,branch and marks\n");
scanf("%d%s%s%f",&s.rno,s.name,s.branch,&s.marks);
printf("%d\t%s\t%s\t%f",p->rno,p->name,p->branch,p->marks);
}
```

5. Explain structures and functions with an example program?

Structures and Functions:-

To pass a copy of entire structure as an actual argument to a function is known as structures and functions.

Example: Define a structure type student that would contain roll number,name,branch and marks. Write a c program to pass a copy of entire structure to a function.

```
#include<stdio.h>
struct student
{
int rno;
char name[20];
char branch[20];
float marks;
};
int display(struct student x);
int main()
{
struct student s;
printf("Enter the student rno,name,branch and marks\n");
scanf("%d%s%s%f",&s.rno,s.name,s.branch,&s.marks);
display(s);
}
display(struct student x)
{
printf("%d\t%s\t%s\t%f",x.rno,x.name,x.branch,x.marks);
}
```

6. Write short notes on the following

1. Nested structures

2. Self referential structures

1. Nested structures:

A structure member maybe a structure is called nested structure or structures with structures.

Example Program:

Example: Define a structure type student that would contain roll number,name,branch,marks and date of birth. Write a C program to read this information from the keyboard and print the same on the monitor using nested structure.

```
#include<stdio.h>
struct date
{
int day;
char month[20];
int year;
};
struct student
{
int rno;
char name[20];
char branch[20];
float marks;
struct date d;
};
int main()
{
struct student s;
printf("Enter the student rno,name,branch,marks,day,month,year\n");
scanf("%d%s%s%f%d%s%d",&s.rno,s.name,s.branch,&s.marks,&s.d.day,s.d.month,&s.d
.year);
printf("the student details are\n");
printf("%d\t%s\t%s\t%f\t%d\t%s\t%d",s.rno,s.name,s.branch,s.marks,s.d.day,s.d.mo
nth,s.d.year);
}
```

2. Self referential structures:

A structure which contains atleast one member as a pointer to the same structure is known as Self referential structures.

Self referential structures are mainly used in the implementation of data structures like linked lists and trees.

Example:

```
struct student
{
char name[20];
int rollno;
struct student *ptr;
};
```

7. Define union and explain about declaration, initialization and accessing of union members in C with an example program?

Union is a collection of elements of different data types.

Declaration of a union:-

Syntax:

```

union tagname
{
datatype member1;
datatype member2;
datatype member3;
.
.
datatype member n;
};

```

In the above syntax union is a keyword that declares/stores the members or fields of a union.

tag name is name of the union.

datatype maybe any basic datatype such as int,float,char etc.

member1,member2,.....,member n are union members or fields of a union.

The body of a union should be end with semicolon(;).

Example:

```

union student
{
int rno;
char name[20];
char branch[20];
float marks;
};

```

Declaration of a union variable:-

Syntax:-

```
union tagname variable;
```

Example:

```
union student s;
```

Initialization of a union variable:-

Syntax:-

```
union tagname variable={member1,member2,...member n};
```

Example:

```
union student s={501,"Dinesh","cse",100.5};
```

Member operator or dot operator(.):-

To access the each member of a union using the union variable we have to use dot operator (.) in c language.

Syntax:-

```
union variable. union member
```

Example Program:

```

#include<stdio.h>
union student
{
int rno;
char name[20];
float marks;
};
int main()
{
union student s;
printf("enter the student rollno\n");
scanf("%d",&s.rno);
printf("the rno=%d",s.rno);
}

```

```
printf("enter the student name\n");
scanf("%s",s.name);
printf("name of the student is %s",s.name);
printf("enter the marks of the student\n");
scanf("%f",&s.marks);
printf("marks of the student is %f",s.marks);
}
```

8. Differentiate between structure and union?

Structure

1. Structure is a collection of elements of different data types.

2. Declaration of a structure:-

```
struct tagname
{
datatype member1;
datatype member2;
datatype member3;
.
.
datatype member n;
};
```

3. struct is a keyword that declares/stores the members or fields of a structure.

4. The body of a structure should be end with semicolon(;).

5. Example:-

```
struct student
{
int rollno;
char name[20];
char branch[20];
float marks;
};
```

6. Declaration of a structure variable:-
struct tagname variable;

Example:

```
struct student s;
```

7. Member operator/Dot operator (.):-

To access the each member of a structure using the structure variable we have to use dot operator (.) in C language.

Syntax:-

```
structurevariable.structuremember
```

8. In structures each member has its own storage allocation.

9. For example

```
struct student
{
int rollno;
char name[20];
```

Union

1. union is a collection of elements of different data types.

2. Declaration of a union:-

```
union tagname
{
datatype member1;
datatype member2;
datatype member3;
.
.
datatype member n;
};
```

3. union is a keyword that declares/stores the members or fields of a union.

4. The body of a union should be end with semicolon(;).

5. Example:-

```
union student
{
int rollno;
char name[20];
char branch[20];
float marks;
};
```

6. Declaration of a union variable:-
union tagname variable;

Example:

```
union student s;
```

7. Member operator/Dot operator (.):-

To access the each member of a union using the union variable we have to use dot operator (.) in C language.

Syntax:-

```
unionvariable.unionmember
```

8. In unions all the members use the same memory allocation i.e., the highest storage allocation.

9. For example

```
union student
{
int rollno;
char name[20];
```

float marks; }; The total memory required to store a structure variable is equal to the sum of size of all the members. In the above case 26 bytes(2+20+4) will be required to store structure variable. 10. Structure can contain any numbers of members and it can handle all the members at the same time.	float marks; }; The total memory required to store a union variable is equal to the member with largest size. In the above case 20 bytes will be required to store union variable. 10. Union can contain any numbers of members but it can handle one member at a time.
---	---

9. Define files and streams and its types and explain how to opening and closing a file?

Files in C

A File is collection of related data stored permanently in computer. Using files we can store our data in Secondary memory (Hard disk).

Types of Files

1. Text files
2. Binary files

1. Text files

Text files are the normal .txt files that you can easily create using notepad or any simple text editors.

When you open those files, you will see all the contents within the file as plain text. You can easily edit or delete the contents.

They take minimum effort to maintain, are easily readable, and provide least security and takes bigger storage space.

2. Binary files

Binary files are mostly the .bin files in your computer.

Instead of storing data in plain text, they store it in the binary form (0's and 1's).

They can hold higher amount of data, are not readable easily and provides a better security than text files.

Streams:

All input and output operations in C is performed with streams. Stream is a sequence of characters organized into lines (text stream), each line consists of 0 or more characters and ends with the newline character '\n'. Streams provide communications channels between files and programs. When program execution begins, three files and their associated streams are connected to the program automatically.

They are

1. the standard input stream(stdin)
2. the standard output stream(stdout)
3. the standard error stream(stderr)

Standard input: It is connected to the keyboard & it enables a program to read data from the keyboard.

Standard output: It is connected to the screen & it enables a program to write data to the screen.

Standard error: It is connected to the screen & all error messages are output to standard error.

File Operations in C:

There are 5 basic operations on files, They are

1. Naming a file
2. Opening a file
3. Reading data from a file
4. Writing data to a file
5. Closing a file

Defining and Opening a File

Data structure of file is defined as FILE in the standard I/O function. So all files should be declared as type FILE.

Declaration/Syntax of a file:

```
FILE *fp;
fp=fopen("filename", "mode");
```

Here fp declare a variable as a pointer to the data type FILE.
In the above syntax mode specifies purpose of opening a file.

File Opening mode

S.No	Mode	Meaning	Purpose
1	r	Reading	Open the file for reading only.
2	w	Writing	Open the file for writing only.
3	a	Appending	Open the file for appending (or adding) data to it.
4	r+	Reading + Writing	New data is written at the beginning override existing data.
5	w+	Writing + Reading	Override existing data.
6	a+	Reading + Appending	To new data is appended at the end of file.

Closing a File

A file must be close after completion of all operations related to a file. For closing a file we need fclose() function.

Syntax:

```
fclose(filepointer);
```

10. Discuss file Handling functions/file input and output functions /sequential access functions in C?

There are 8 types of file handling functions:

- 1) fopen()
- 2) fclose()
- 3) getc()/fgetc()
- 4) putc()/fputc()

- 5) `getw()`
- 6) `putw()`
- 7) `fscanf()`
- 8) `fprintf()`

1) `fopen()`: This can be used to create a new file or open an existing file.

Syntax: `fp=fopen("filename","mode");`

2) `fclose()`: A file must be closed after all operations completed on it.

Syntax: `fclose(fp);`

3) `getc()/fgetc()`: This function can be used to read a character from a file.

Syntax: `c=getc(fp);`

4) `putc()/fputc()`: This function can be used to write a character to a file.

Syntax: `putc(c,fp);`

5) `getw()`: This can be used to read an integer from a file.

Syntax: `n=getw(fp);`

6) `putw()`: This can be used to write an integer to a file.

Syntax: `putw(n,fp);`

7) `fscanf()`: This can be used to read a set of data values from a file.

Syntax: `fscanf(fp,"control string",&variablelist);`

8) `fprintf()`: This can be used to write a set of data values to a file.

Syntax: `fprintf(fp,"control string",variablelist);`

11. Explain file position functions / random access file functions in C?

File Positioning Functions/Random Access File Functions

1. fseek()

This function can be used to sets the position to a desired point in the file.

Syntax:

`fseek(fp,offset,position);`

2. ftell()

This function can be used to gives the current position in the file (in terms of bytes from the start).

Syntax:

```
n=ftell(fp);
```

3. rewind()

This function can be used to sets the position to the beginning of the file.

Syntax:

```
rewind(fp);
```

12. Write a C program to read the data from the keyboard, write it to a file called "INPUT" and again read the same data from that file and print it on monitor?

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

```
int main( )
```

```
{
```

```
FILE *f1;
```

```
char ch;
```

```
f1=fopen("INPUT","w");
```

```
printf("Enter the data to the INPUT file and press ctrl Z to stop\n");
```

```
while((ch=getchar())!=EOF)
```

```
{
```

```
putc(ch,f1);
```

```
}
```

```
fclose(f1);
```

```
f1=fopen("INPUT","r");
```

```
while((ch=getc(fp))!=EOF)
```

```
{
```

```
putchar(ch);
```

```
}
```

```
fclose(f1);
```

```
}
```

13. Write a C program to copy the contents of one file into another file?

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

```
int main( )
```

```
{
```

```
FILE *f1,*f2;
```

```
char ch;
```

```
f1=fopen("source.txt","w");
printf("Enter the data to the source.txt file and press ctrl Z to stop\n");
while((ch=getchar())!=EOF)
{
    putc(ch,f1);
}
fclose(f1);
f1=fopen("source.txt","r");
f2=fopen("target.txt","w");
while((ch=getc(f1))!=EOF)
{
    putc(ch,f2);
}
fclose(f1);
fclose(f2);
printf("After copying the data in target.txt file is\n");
f2=fopen("target.txt","r");
while((ch=getc(f2))!=EOF)
{
    putchar(ch);
}
fclose(f2);
}
```

14. Write a C program to merge the contents of two files into third file?

```
#include<stdio.h>

int main( )
{
    FILE *f1,*f2,f3;
    char ch;
    f1=fopen("file1.txt","w");
    printf("Enter the data to the file1.txt file and press ctrl Z to stop\n");
    while((ch=getchar())!=EOF)
```

```
{
putc(ch,f1);
}
fclose(f1);
f1=fopen("file1.txt","r");
f3=fopen("file3.txt","w");
while((ch=getc(f1))!=EOF)
{
putc(ch,f3);
}
fclose(f1);
fclose(f3);
f2=fopen("file2.txt","w");
printf("Enter the data to the file2.txt file and press ctrl Z to stop\n");
while((ch=getchar())!=EOF)
{
putc(ch,f2);
}
fclose(f2);
f2=fopen("file2.txt","r");
f3=fopen("file3.txt","w");
while((ch=getc(f2))!=EOF)
{
putc(ch,f3);
}
fclose(f2);
fclose(f3);
printf("After merging two files into third file is\n");
f3=fopen("file3.txt","r");
while((ch=getc(f3))!=EOF)
{
putchar(ch);
```

```
}  
fclose(f3);  
}
```

UNIT-5

1. Define data structure and its types?

Data structure is a method of organizing a large amount of data more efficiently so that any operation on that data becomes easy.

Based on the organizing method of data structure, data structures are divided into two types.

Linear Data Structures

Non - Linear Data Structures

Linear Data Structures

If a data structure organizes the data in sequential order, then that data structure is called a Linear Data Structure.

Examples

Arrays

List (Linked List)

Stack

Queue

Non - Linear Data Structures

If a data structure organizes the data in random order, then that data structure is called as Non-Linear Data Structure.

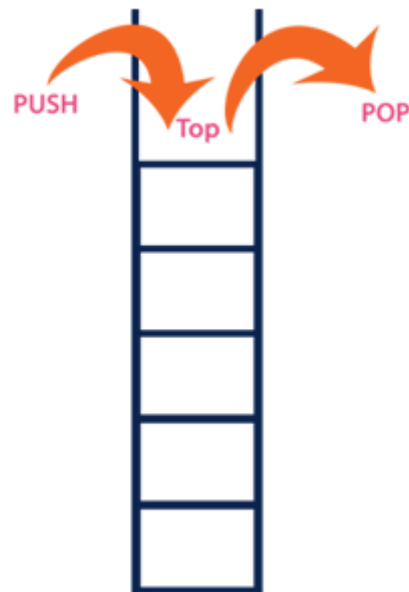
Examples

Trees

Graphs

2. Define stack and its operations?

Stack is a linear data structure in which the insertion and deletion operations are performed at only one end. In a stack, adding and removing of elements are performed at a single position which is known as "top". That means, a new element is added at top of the stack and an element is removed from the top of the stack. In stack, the insertion and deletion operations are performed based on LIFO (Last In First Out) principle.



In a stack, the insertion operation is performed using a function called "push" and deletion operation is performed using a function called "pop".

In the figure, PUSH and POP operations are performed at a top position in the stack. That means, both the insertion and deletion operations are performed at one end (i.e., at Top)

Example

If we want to create a stack by inserting 10,45,12,16,35 and 50. Then 10 becomes the bottom-most element and 50 is the topmost element. The last inserted element 50 is at Top of the stack as shown in the image below.



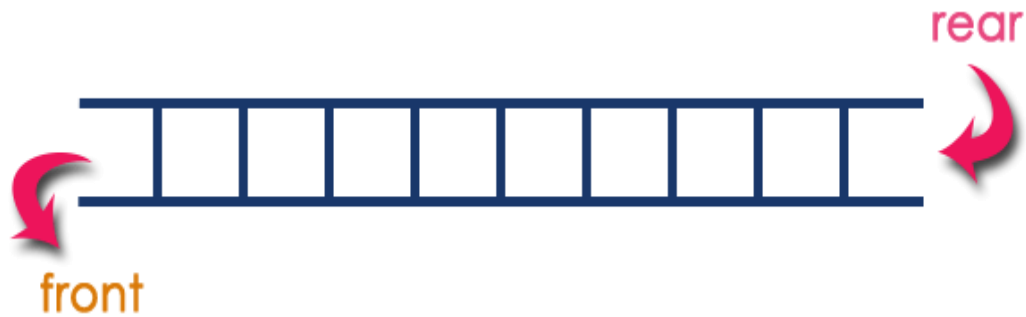
Operations on a Stack

The following operations are performed on the stack.

1. Push (To insert an element on to the stack)
2. Pop (To delete an element from the stack)
3. Display (To display elements of the stack)

3. Define queue and its operations?

Queue is a linear data structure in which the insertion and deletion operations are performed at two different ends. In a queue data structure, adding and removing elements are performed at two different positions. The insertion is performed at one end and deletion is performed at another end. In a queue data structure, the insertion operation is performed at a position which is known as 'rear' and the deletion operation is performed at a position which is known as 'front'. In queue data structure, the insertion and deletion operations are performed based on FIFO (First In First Out) principle.



Example

Queue after inserting 25, 30, 51, 60 and 85.

After Inserting five elements...



Operations on a Queue

The following operations are performed on the stack.

1. Rear (To insert an element on to the queue)
2. Front (To delete an element from the queue)
3. Display (To display elements of the queue)

4. Write a C program to implement stack and its operations using array representation?

```
#include<stdio.h>
int stack[100],choice,n,top=-1,x,i;
int push();
int pop();
int display();
int main()
{
    printf("Enter the size of stack:\n");
    scanf("%d",&n);
```

```
printf("1.PUSH\n2.POP\n3.DISPLAY\n4.EXIT\n");
do
{
    printf("Enter your Choice:\n");
    scanf("%d",&choice);
    switch(choice)
    {
        case 1:
        {
            push();
            break;
        }
        case 2:
        {
            pop();
            break;
        }
        case 3:
        {
            display();
            break;
        }
        case 4:
        {
            printf("EXIT POINT\n");
            break;
        }
        default:
        {
            printf("Please Enter a Valid Choice(1/2/3/4)\n");
        }
    }
}
while(choice!=4);
}
int push()
{
    if(top>=n-1)
    {
        printf("stack is over flow\n");
    }
    else
    {
        printf("Enter a value to be pushed:\n");
        scanf("%d",&x);
        top++;
        stack[top]=x;
    }
}
```

```

    }
}
int pop()
{
    if(top<=-1)
    {
        printf("stack is under flow\n");
    }
    else
    {
        printf("The popped elements is %d\n",stack[top]);
        top--;
    }
}
int display()
{
    if(top>=0)
    {
        printf("The elements in stack\n");
        for(i=top;i>=0;i--)
        {
            printf("%d\n",stack[i]);
        }
    }
    else
    {
        printf("The stack is empty\n");
    }
}

```

5. Write a C program to implement queue and its operations using array representation?

```

#include<stdio.h>
int queue[100],choice,n,front=-1,rear=-1,x,i;
int insertion();
int deletion();
int display();
int main()
{
    printf("Enter the size of queue:\n");
    scanf("%d",&n);
    printf("1.Insertion\n2.Deletion\n3.Display\n4.Exit\n");
    do
    {
        printf("Enter your Choice:\n");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:

```



```
        {
            insertion();
            break;
        }
        case 2:
        {
            deletion();
            break;
        }
        case 3:
        {
            display();
            break;
        }
        case 4:
        {
            printf("EXIT POINT\n");
            break;
        }
        default:
        {
            printf("Please Enter a Valid Choice(1/2/3/4)\n");
        }
    }
}
while(choice!=4);
}
int insertion()
{
    if(rear==n-1)
        printf("Queue Overflow\n");
    else
    {
        if(front==-1)
            front=0;
        printf("Insert the element in queue:\n");
        scanf("%d",&x);
        rear=rear+1;
        queue[rear]=x;
    }
}
int deletion()
{
    if(front==-1||front>rear)
    {
        printf("Queue Underflow\n");
    }
    else
    {

```

```
        printf("Element deleted from queue is:%d\n",queue[front]);
        front=front+1;
    }
}
int display()
{
    if(front==-1)
    {
        printf("Queue is empty\n");
    }
    else
    {
        printf("The elements in queue\n");
        for(i=front;i<=rear;i++)
        {
            printf("%d\n",queue[i]);
        }
    }
}
```