MODULE: 2
SE – Introduction to Programming [C Program]

1. **Overview of C Programming :- Write an essay covering the history and evolution of C programming. Explain its importance and why it is still used today.**

- **History** of C language is interesting to know. Here we are going to discuss a brief history of the c language.
- ~ C programming language was developed in 1972 by Dennis Ritchie at bell laboratories of AT&T (American Telephone & Telegraph), located in the U.S.A.
- ~ Dennis Ritchie is known as the founder of the c language.
- ~ It was developed to overcome the problems of previous languages such as B, BCPL, etc.
- ~ Initially, C language was developed to be used in UNIX operating system. It inherits many features of previous languages such as B and BCPL.
- ~ Let's see the programming languages that were developed before C language.

LANGUAGE	**YEAR**	**DEVELOPED BY**
Algol	1960	International Group
BCPL	1967	Martin Richard
B	1970	Ken Thompson
Traditional C	1972	Dennis Ritchie
K & R C	1978	Kernighan & Dennis Ritchie
ANSI C	1989	ANSI Committee
ANSI/ISO	1990	ISO Committee
C99	1999	Standardization Committee

- **Importance:-** C is called as a robust language, which has so many built-in functions and operations, which can be used to write any complex program.
- ~ The point on C's ability to extend itself could be expanded by mentioning that C supports libraries, which enables code reuse and the addition of custom functions.
- ~ 'C' language is best for structured programming, where the user can think of a problem in terms of function modules (or) blocks.
- ~ The C language is essential in programming due to its efficiency, control over hardware, and widespread use in foundational software development.
- ~ Its low-level capabilities allow direct manipulation of memory, which makes it ideal for system programming—especially in operating systems, device drivers, and embedded systems where performance is critical.
- **Used:-** The reasons provided are accurate: C's speed, portability, and influence on modern languages are well-explained.
- ~ It provides a minimal set of commands with a powerful structure, enabling it to perform optimally across various hardware platforms. Operating systems like Windows, Linux, and macOS are largely built in C, as are databases and even many parts of modern programming languages like Python.
- ~ C's simplicity and robust functionality make it a perfect language for understanding core programming concepts, which is why it's also popular in education.
- ~ Additionally, many high-level languages are built on C or inherit its syntax and structure, making knowledge of C foundational for learning other languages like C++, Java, and even Python.

2. **Setting Up Environment:- Describe the steps to install a C compiler (e.g., GCC) and set up an Integrated Development Environment (IDE) like DevC++, VS Code, or CodeBlocks.**

- **Steps to Set Up GCC Compiler for Windows:-**

~ Download MinGW :- Search "MinGW C Compiler" or visit SourceForge.net to download.
~ Run the Installer:- Open the setup file from your Downloads folder and start the installation.
~ Choose Installation Directory:- Select the installation location (default is usually best), then proceed with the installation.
~ Install Required Packages:- Open MinGW Installation Manager and mark packages like `mingw32-base` (for C), `mingw32-gcc-g++` (for C++), and others if needed.
~ Apply and Download Changes:- Go to `Installation > Apply Changes` to download and install selected packages.
~ Add GCC to System Path:- Go to System Properties > Environment Variables and add the `bin` directory of your MinGW or MSYS2 installation to the Path variable (e.g., `C:\MinGW\bin` or `C:\msys64\mingw64\bin`).
~ Verify Installation:- Open Command Prompt and type `gcc --version` to check if GCC is installed and working correctly.
~ This completes the GCC setup on Windows, enabling C and C++ compilation.

- **Steps to Set Up Visual Studio Code (VS Code) as an IDE:-**

~ Step 1: Visit the Official Website of the Visual Studio Code using any web browser like Google Chrome, Microsoft Edge, etc.
~ Step 2: Press the “Download for Windows” button on the website to start the download of the Visual Studio Code Application.
~ Step 3: When the download finishes, then the Visual Studio Code Icon appears in the downloads folder.
~ Step 4: Click on the Installer icon to start the installation process of the Visual Studio Code.
~ Step 5: After the Installer opens, it will ask you to accept the terms and conditions of the Visual Studio Code. Click on I accept the agreement and then click the Next button.
~ Step 6: Choose the location data for running the Visual Studio Code. It will then ask you to browse the location. Then click on the Next button.
~ Step 7: Then it will ask to begin the installation setup. Click on the Install button.
~ Step 8: After clicking on Install, it will take about 1 minute to install the Visual Studio Code on your device.
~ Step 9: After the Installation setup for Visual Studio Code is finished, it will show a window like this below. Tick the “Launch Visual Studio Code” checkbox and then click Next.
~ Step 10: After the previous step, the Visual Studio Code window opens successfully. Now you can create a new file in the Visual Studio Code window and choose a language of yours to begin your programming journey.

3. **Basic Structure of a C Program:- Explain the basic structure of a C program, including headers, main function, comments, data types, and variables. Provide examples.**

➢ **Structure of a C program**

- Documentation:- This section consists of the description of the program, the name of program, and the creation date and time of the program. It is specified at the start of the program in the form of comments. Basically, it gives an overview to the reader of the programmer.
- Comments are used to explain the code. They are ignored by the compiler and do not affect program execution. In C, there are two types of comments:
  Single-line comments: Start with // and continue until the end of the line.
  Multi-line comments: Enclosed between /* and */.
- Preprocessor Section/Header Files:- All the header files of the program will be declared in beginning of program. Header files help us to access other's improved code into our code. A copy of these multiple file inserted into our program before the process of compilation.

~ Example:- #include<stdio.h>   #include<math.h>

- Main() Function:- Every C program must have a main function. The main() function of the program is written in this section. Operations like declaration and execution are performed inside the curly braces of the main program. The return type of the main() function can be int as well as void too. Example:- void main()   int main()
- Variables:- Variables are used to store data that can be used and manipulated in the program. Variables must be declared with a specific data type before they can be used.

~ Example:- int age = 20;   // age is variable

- Global Declaration:- The global declaration section contains global variables, function declaration, and static variables. Variables and functions which are declared in this scope can be used anywhere in the program. Example:- int num = 18;
- Data Types:- It define the type of data a variable can store. Common data types are:

~ int: Integer type (e.g., int x = 5;).
~ float: Floating-point type (e.g., float y = 3.14;).
~ char: Character type (e.g., char ch = 'A';).
~ double: Double-precision floating-point type (e.g., double d = 3.14159;).

- **Example:-**

```
#include <stdio.h>   // Include the standard input/output library
// Main function: Execution starts here
int main()
{   // Declare variables
    int age = 25;               // Integer variable to store age
    float height = 5.9;           // Float variable to store height
    char initial = 'J';          // Char variable to store first letter of name

    // Print values to the console
    printf("Age: %d\n", age);      // Display the age
    printf("Height: %.2f\n", height); // Display the height with 2 decimal places
    printf("Initial: %c\n", initial); // Display the initial

    return 0;  // Exit the program successfully
}
```

4. **Operators in C:- Write notes explaining each type of operator in C: arithmetic, relational, logical, assignment, increment/decrement, bitwise, and conditional operators.**

- **Operator** in C can be defined as the symbol that help us to perform some specific mathematical, relational, bitwise, conditional, or logical computations on values and variables. The values and variables used with operators are called operands. So we can say that the operators are the symbols that perform operations on operands.
- **Types of Operators in C**

- Arithmetic Operators:- The arithmetic operators are used to perform arithmetic/mathematical operations on operands.

SYMBOL	OPERATOR	DESCRIPTION	SYNTAX
+	Plus	Adds two numeric values.	A + B
-	Minus	Subtracts right operand from left operand.	A - B
*	Multiply	Multiply two numeric values.	A * B
/	Divide	Divide two numeric values.	A / B
%	Modulo	Returns the remainder after diving the left operand with the right operand.	A % B
+	Unary Plus	Used to specify the positive values.	+A
-	Unary Minus	Flips the sign of the value.	-A
++	Increment	Increases the value of the operand by 1.	A++, ++A
--	Decrement	Decreases the value of the operand by 1.	A--, --A

- Relational Operators:- The relational operators in C are used for the comparison of the two operands. All these operators are binary operators that return true or false values as the result of comparison.

SYMBOL	OPERATOR	DESCRIPTION	SYNTAX
<	Less Than	Returns true if the left operand is less than the right operand. Else false	A < B
>	Greater Than	Returns true if the left operand is greater than the right operand. Else false	A > B
<=	Less Than or Equal to	Returns true if the left operand is less than or equal to the right operand. Else false	A <= B
>=	Greater Than or Equal to	Returns true if the left operand is greater than or equal to right operand. Else false	A >= B
==	Equal To	Returns true if both the operands are equal.	A == B
!=	Not Equal To	Returns true if both the operands are NOT equal.	A != B

- Logical Operators:- Logical Operators are used to combine two or more conditions/constraints or to complement the evaluation of the original condition in consideration. The result of the operation of a logical operator is a Boolean value either true or false.

SYMBOL	OPERATOR	DESCRIPTION	SYNTAX
&&	Logical AND	Returns true if both the operands are true.	A && B
||	Logical OR	Returns true if both or any of the operand is true.	A || B

!	Logical NOT	Returns true if the operand is false.	A ! B

- <u>Bitwise Operators:-</u> The Bitwise operators are used to perform bit-level operations on the operands. The operators are first converted to bit-level and then the calculation is performed on the operands. Mathematical operations such as addition, subtraction, multiplication, etc. can be performed at the bit level for faster processing.

SYMBOL	OPERATOR	DESCRIPTION	SYNTAX
&	Bitwise AND	Performs bit-by-bit AND operation and returns the result.	A & B
|	Bitwise OR	Performs bit-by-bit OR operation and returns the result.	A | B
^	Bitwise XOR	Performs bit-by-bit XOR operation and returns the result.	A ^ B
~	Bitwise First Complement	Flips all the set and unset bits on the number.	~A
<<	Bitwise LeftShift	Shifts the number in binary form by one place in the operation and returns the result.	A << B
>>	Bitwise RightShift	Shifts the number in binary form by one place in the operation and returns the result.	A >> B

- <u>Assignment Operators:-</u> Used to assign values to variable.

SYMBOL	OPERATOR	DESCRIPTION	SYNTAX
=	**Simple Assign**	**Assign the value of the right operand to the left operand.**	**A = B**
+=	Plus and Assign	Add the right operand and left operand and assign this value to the left operand.	A += B
-=	Minus and Assign	Subtract the right operand and left operand and assign this value to the left operand.	A -= B
*=	Multiply and Assign	Multiply the right operand and left operand and assign this value to the left operand.	A *= B
/=	Divide and Assign	Divide the left operand with the right operand and assign this value to the left operand.	A /= B
%=	Modulo and Assign	Assign the remainder in the division of left operand with the right operand to the left operand.	A %= B
&=	AND and Assign	Performs bitwise AND and assigns this value to the left operand.	A &= B
|=	OR and Assign	Performs bitwise OR and assigns this value to the left operand.	A |= B
^=	XOR and Assign	Performs bitwise XOR and assigns this value to the left operand.	A ^= B
<<=	LefShift and Assign	Performs bitwise Leftshift and assign this value to the left operand.	A <<= B
>>=	RightShift and Assign	Performs bitwise Rightshift and assign this value to the left operand.	A >>=B

- <u>Other Operators:-</u>

~ sizeof Operator:- Returns the size of a variable or data type in bytes.
  Syntax:- sizeof (operand)

~ Conditional Operator:- We may replace the use of if…else statements with conditional operator.
  Synatx:- (Condition) ? (Expresion1-TRUE) : (Expression2-FALSE) ;
  Here, Condition to be evaluated. If the condition is True then we will execute and return the result of Expression1 otherwise if the condition is false then we will execute and return the result of Expression2.

~ Type Casting Operator:- Convert one Data type to another.
  Syntax:- (new_type) operands;

5. Control Flow Statements in C:- Explain decision-making statements in C (if, else, nested if-else, switch). Provide examples of each.

➢ **Decision-making statement:-** They are also known as conditional statements and are used to evaluate one or more conditions and make the decision whether to execute a set of statements or not. These decision-making statements in programming languages decide the direction of the flow of program execution.

➢ **Types of Conditional Statements:-**

- if Statement:- The simplest decision-making statement is the if statement. It evaluates a condition, and if the condition is true, it executes the code inside the block. If the condition is false, the block is skipped.

~ Syntax:-
```
if(condition)
{    // Statements to execute if condition is true     }
```
~ Example:-
```
#include <stdio.h>
int main()
{
int x = 10;
if (x > 5)
{
printf("x is greater than 5\n");
}
}
```
Output:- x is greater than 5

- if-else Statement:- The if-else statement allows for two paths of execution. If the condition is true, the if block executes; otherwise, the else block executes.

~ Syntax:-
```
if (condition)
{    // Code to execute if condition is true      }
else
{    // Code to execute if condition is false     }
```
~ Example:-
```
#include <stdio.h>
int main()
{
int x = 4;
if (x > 5)
{
printf("x is greater than 5\n");
}
else
{
printf("x is less than 5\n");
}
}
Output: x is less than 5
```

- <u>if-else-if Ladder:-</u> The if-else-if ladder is used when there are multiple conditions to check, and each condition is mutually exclusive. It allows the program to choose one of many possible code blocks based on the conditions.

~ Syntax:-

```
if(condition)
    Statement ;
elseif (condition)
    Statement ;
elseif(condition)
    Statement ;
.. else
    Statement ;
```

~ Example:-

```
#include <stdio.h>
int main()
{
int Num;
printf("Enter number:");
scanf("%d",&Num);
if (Num == 10)
printf("Number is 10");
else if (Num == 15)
printf("Number is 15");
else if (Num == 20)
printf("Number is 20");
else
printf("Number is not present");
}
Output:- Enter number:20
Number is 20
```

- <u>Nested if Statement:-</u> A nested if statement is an if statement inside another if statement. It allows multiple conditions to be checked sequentially, providing more detailed decision-making.

~ Syntax:-

```
if (condition1)
{    // Executes when condition1 is true
if (condition_2)
{    // statement 1        }
else
{    // Statement 2        }
}
else
```

```
{
if (condition_3)
{    // statement 3         }
else
{    // Statement 4         }
}
```

~ Example:- // program to check weather male or female is eligible for marriage or not.

```
#include<stdio.h>
int main()
{
int age;
char ch;
printf("\n M ||m for male.\n F || f for female");
printf("Enter your choice:");
scanf("%c",&ch);
if(ch=='m' || ch== 'M')
{
printf("Enter age:");
scanf("%d",&age);
if(age>=21)
{
printf("Person is eligible for marriage");
}
else
{
printf("person is not eligible for marriage.");
}
}
else if (ch=='f'|| ch=='F')
{
printf("Enter age:");
scanf("%d",&age);
if(age>=18)
{
printf("Person is eligible for marriage");
}
else
{
printf("person is not eligible for marriage.");
}
}
else
{
printf("Invalid choice...");
}
}
Output:-
M ||m for male.
F || f for female
Enter your choice:  f
Enter age:12
person is not eligible for marriage.
```

- switch Statement:- The switch statement is used when you have multiple possible conditions based on the value of a single variable. Unlike the if-else-if ladder, the switch statement is cleaner and more efficient when there are many cases. The switch block consists of cases to be executed based on the value of the switch variable.

~ Syntax:-

```
switch (expression)
{     case value1:
statements;
break;
case value2:
statements;
break;
....     ....     ....
default:
statements;         }
```

~ Example:-

```
#include <stdio.h>
int main()
{
int var = 2;
switch (var)
{
case 1:
printf("Case 1 is executed");
break;
case 2:
printf("Case 2 is executed");
break;
default:
printf("Default Case is executed");
break;
}
}
Output:- Case 2 is executed
```

- Nested Switch:- A nested switch occurs when one switch statement is inside another. This can be useful for checking multiple levels of conditions.

~ Syntax:-

```
switch (expression1)
{ case value1:
switch (expression2)
{    case value2:
     break;          }
break;
}
```

~ Example:-

```
#include <stdio.h>
int main()
{
int x = 1, y = 2;
switch (x)
{
case 1:
switch (y)
{
case 2:
printf("x is 1 and y is 2\n");
break;
default:
printf("y is not 2\n");
}
break;
default:
printf("x is not 1\n");
}
}
```

Output:- x is 1 and y is 2

6. Looping in C:- Compare and contrast while loops, for loops, and do-while loops. Explain the scenarios in which each loop is most appropriate.

- **Loops** in programming are used to repeat a block of code until the specified condition is met. A loop statement allows programmers to execute a statement or group of statements multiple times without repetition of code.
- **Types of Loop:-**

- For Loop:- It is a repetition control structure that allows programmers to write a loop that will be executed a specific number of times. for loop enables programmers to perform n number of steps together in a single line.

~ Syntax:-

```
for(initialization ;Condition; increment/decrement)
{    // body of loop;  }
```

~ Example:-

```
#include<stdio.h>
int main()
{
int i;
for(i=1;i<=5;i++)
{
printf("%d \t",i);
}
}
Output:- 1          2        3        4        5
```

- While Loop:- The while loop repeats a block of code as long as a given condition is true. It is called an entry-controlled loop because it checks the condition at the beginning, meaning the loop may not execute at all if the condition is initially false.

~ Syntax:-

```
initialization;
while(Condition)
{    body of loop;
increment/decrement;  }
```

~ Example:-

```
#include<stdio.h>
int main()
{
int i=1;
int num;
printf("Enter number:");
scanf("%d",&num);
while(i<=num)
{
printf("%d \t ",i);
i++;
}
}
Output:- Enter number: 4
1   2        3        4
```

- <u>Do While Loop:-</u> The do-while loop is a post-controlled loop, meaning it checks the condition at the end of each iteration. This ensures that the code block executes at least once, regardless of the condition's initial value.

~ Syntax:-

```
initialization;
do
{    body of loop;
increment/decrement
}
while(Condition);
```

~ Example:-

```
#include<stdio.h>
int main()
{
int num=10;
do
{
printf("%d\t",num);
num--;
}
while(num>=0);
}
Output:- 10       9       8       7       6       5       4       3       2       1       0
```

7. **Loop Control Statements:- Explain the use of break, continue, and goto statements in C. Provide examples of each.**

- **loop control statements** manage the flow within loops, allowing you to jump out of loops, skip certain iterations, or redirect the flow.
- **Types of Loop Control Statement:-**

- Control/Break Statement:- The break statement is used to terminate the switch and loop statement. It transfers the execution to the statement immediately following the loop or switch.

~ Syntax:- break;

~ Example:-

```
#include <stdio.h>
int main()
{
for (int i = 1; i <= 10; i++)
{
if (i == 5)
{
break; // Exit the loop when i is 5
}
printf("%d\t", i);
}
}
```

Output:-
```
1	2	3	4
```

- Continue Statement:- Continue statement skips the remainder body and immediately resets its condition before reiterating it. To skip an iteration when a certain condition is met.

~ Syntax:- continue;

~ Example:-

```
#include <stdio.h>
int main()
{
for (int i = 1; i <= 10; i++)
{
if (i == 5)
{
continue; // Skip printing when i is 5
}
printf("%d\t", i);
}
}
```

Output:-
```
1 2	3	4	6	7	8	9	10
```

- Goto Statement:- Goto statement transfers the control to the labeled statement. To break out of multiple nested loops or to handle exceptional cases.

~ Syntax:-

```
goto label; // Code here is skipped
```

```
label: // Code execution resumes here
```

~ Example:-

```
#include <stdio.h>
int main()
{
int x = 1;
if (x == 1)
{
goto skip; // Jump to the label "skip"
}
printf("This will not print because of goto.\n");
skip:
printf("This is where execution jumps to with goto.\n");
}
```

Output:- This is where execution jumps to with goto.

- <u>Return Statement:-</u> The return statement is used to exit a function and return a value to the caller (if specified). While technically not a loop control statement, return can control program flow within loops and functions. To terminate function execution and (optionally) return a value. Often used for early exits from functions in case of an error or a certain condition.

~ Syntax:-

```
return; // in void functions
return value; // in functions with a return type
```

~ Example:-

```
#include <stdio.h>
int main()
{
for (int i = 1; i <= 10; i++)
{
if (i == 5)
{ return 0; // Exit the main function when i is 5
}
printf("%d\t", i);
}
return 0;
}
```

Output:-

```
1	2	3	4
```

8. Functions in C:- What are functions in C? Explain function declaration, definition, and how to call a function. Provide examples.

➢ A **function** is a set of statements that performs a specific task when called. Functions provide modularity and code reusability in a C program. Functions in C are also referred to as subroutines or procedures in other languages. They allow you to break down complex programs into smaller, manageable parts.

➢ **There are three main components of a function in C:**

- Function Declaration (or Prototype):- A function declaration informs the compiler of the function's name, return type, and parameters. It is placed at the beginning of the program, before the main() function, or in a separate header file.

~ int add(int a, int b); tells the compiler that a function named add exists and will return an int.

~ Syntax:- return_type function_name(parameter_type1, parameter_type2);

~ Example:- int add(int a, int b); // Declaration of a function named "add"

- Function Definition:- The function definition contains the actual code or logic for the function. This is where you write what the function does. It includes the actual code to execute when the function is called.

~ The function body int add(int a, int b) {...} defines the logic to add two numbers.

~ Syntax:-

```
return_type function_name(parameter_type parameter_name, ...)
{
// Code that performs the function's task
return value; // Return statement for functions with a return type
}
```

~ Example:-

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

- Function Call:- To execute a function, you need to call it from another part of your program. A function call passes control to the function. A function call executes the function. The program control is transferred to the function's code when called.

~ result = add(num1, num2); calls the add function with num1 and num2 as arguments and stores the result in result.

~ Syntax:- function_name(arguments);

~ Example:- int result = sum(10, 30); // Calls the function "sum" with arguments 10 and 30

- Example:-

```
#include <stdio.h>
// Function declaration (prototype)
int add(int a, int b);
int main()
{
```

```
int num1 = 5,
num2 = 10;
int result; // Function call
result = add(num1, num2);
printf("The sum is: %d\n", result);
return 0;
} // Function definition
int add(int a, int b)
{
int sum = a + b;
return sum;
}
Output:- The sum is:15
```

9. **Arrays in C:- Explain the concept of arrays in C. Differentiate between one-dimensional and multi-dimensional arrays with examples.**

➢ An **array** in C is a collection of variables of the same data type stored in contiguous memory locations. Arrays allow you to store multiple values of the same type under a single variable name and access them using an index. It can be used to store the collection of primitive data types such as int, char, float, etc., and also derived and user-defined data types such as pointers, structures, etc.

➢ **There are mainly three types of the array:**

- <u>One Dimensional (1D) Array:-</u> A one-dimensional array is a list of elements of the same data type. It is useful for storing a sequence of values, such as a list of numbers or characters. Each element can be accessed by its index, with the first element at index 0.

~ Syntax:- data_type array_name[size];

~ Example:-

```
#include<stdio.h>
int main()
{
int arr[4]={11,1,5,17};  // integer array.
for(int i=0;i<7;i++)
{
printf("arr[%d]=%d\n",i,arr[i]);
}
}
```

Output:-

```
arr[0]=11
arr[1]=1
arr[2]=5
arr[3]=17
```

- <u>Two Dimension (2D) Array:-</u> It is an array of arrays, often used to represent tabular data or matrices. Each element is accessed by specifying two indices: row and column.

~ Syntax:- data_type array_name [rows][columns];

~ Example:-

```
#include<stdio.h>
int main()
{
int arr[2][3]={{1,2,3},
               {4,5,6}         };
printf("Printing array:\n");
for(int row=0;row<2;row++)
{
printf("\n");
for(int col=0;col<3;col++)
{
printf("%d\t",arr[row][col]);
}   } }
```

```
Output:- Printing Array:
1   2       3
4   5       6
```

- <u>Multi Dimensional Array:-</u> A three-dimensional array is an array of 2D arrays. This type of array can represent a collection of matrices, or a 3D grid, where each element is accessed by three indices.

~ Syntax:- data_type arr_name[table][row[column];

~ Example:-

```
#include <stdio.h>
int main()
{
int arr[2][3][3] = {       {{1, 2, 3},        {4, 5, 6},        {7, 8, 9}},
                            {{22, 23, 33},        {44, 55, 66},        {77, 88, 99}}};
int row, col, table;
printf("printing array:-");
for (table = 0; table < 3; table++)
{
printf("\n");
for (row = 0; row < 3; row++)
{
printf("\n");
for (col = 0; col < 3; col++)
{
printf("%d ", arr[table][row][col]);
}
}
}
}
```

```
Output:- printing array:
1   2       3
4   5       6
7   8       9

22          23      33
44          55      66
77          88      99
```

10. Pointers in C:- Explain what pointers are in C and how they are declared and initialized. Why are pointers important in C?

- A **pointer** is defined as a derived data type that can store the address of other C variables or a memory location. We can access and manipulate the data stored in that memory location using pointers. Pointers are one of the most powerful features in C, providing direct access to memory, which allows for efficient and flexible handling of data.
- **How to use Pointer:-**

- <u>Pointer Declaration:-</u> In pointer declaration, we only declare the pointer but do not initialize it. To declare a pointer, we use the asterisk (*) dereference operator before its name.

~ Syntax:- datatype * pointer_name;

~ Example:-

```
int *p; // Pointer to an integer
float *fp; // Pointer to a float
char *cp; // Pointer to a char
```

- <u>Pointer Initialization:- It</u> is the process where we assign some initial value to the pointer variable. We generally use the (&: ampersand) address of operator to get the memory address of a variable and then store it in the pointer variable.

~ It is recommended that the pointers should always be initialized to some value before starting using it. Otherwise, it may lead to number of errors.

~ Example:-

```
int var = 10;
int * ptr;
ptr = &var;
OR [single step]
int *ptr = &var;
```

- <u>Pointer Dereferencing:-</u> Dereferencing a pointer is the process of accessing the value stored in the memory address specified in the pointer. To access the value at the address a pointer holds, use the dereference operator (*).

~ Example:- printf("%d", *p); // Outputs 10, the value stored in num

- **Importance of Pointer:-**

- Memory Efficiency: Pointers allow C programs to work directly with memory, making it easier to manage resources effectively, especially when working with large data structures.
- Dynamic Memory Allocation: Pointers enable dynamic memory management through functions like malloc, calloc, and free, allowing memory to be allocated or freed during runtime.
- Efficient Array and String Handling: Pointers make it easier to work with arrays, strings, and large data sets without needing to copy data.
- Function Argument Passing: Pointers allow for passing variables by reference, enabling functions to modify arguments directly and improving efficiency.

11. **Strings in C:- Explain string handling functions like strlen(), strcpy(), strcat(), strcmp(), and strchr(). Provide examples of when these functions are useful.**

- ➢ The C **string** functions are built-in functions that can be used for various operations and manipulations on strings. These string functions can be used to perform tasks such as string copy, concatenation, comparison, length, etc. The <string.h> header file contains these string functions.
- ➢ **Commonly used String Functions:-**

- <u>strcat() String Concatenation:-</u> The strcat() function appends (concatenates) one string to the end of another. It will append a copy of the source string to the end of the destination string. It is useful for combining multiple strings, like creating a full name by concatenating a first and last name.

~ Syntax:- char* strcat(char* dest, const char* src);

- <u>strncat():-</u> This function is used for string handling. This function appends not more than n characters from the string pointed to by **src** to the end of the string pointed to by dest plus a terminating Null-character. It is useful for combining multiple strings, like creating a full name by concatenating a first and last name.

~ Syntax:- char* strncat(char* dest, const char* src, size_t n);

- <u>strlen() String Length:-</u> It calculates the length of a given string. It doesn't count the null character '\0'. It is useful when you need the length of a string for operations like loops, memory allocation, or comparison.

~ Syntax:- int strlen(const char *str);

- <u>strcmp():-</u> The strcmp() function compares two strings lexicographically.

~ Returns 0 if both strings are identical.
~ Returns a negative value if the first string is lexicographically less than the second.
~ Returns a positive value if the first string is greater than the second.
~ It is useful for checking equality or sorting strings, for instance, in alphabetical ordering.
~ Syntax:- int strcmp(const char *str1, const char *str2);

- <u>strcpy() String Copy:-</u> The strcpy() function copies the contents of one string into another. It is useful when you need to copy the content of one string to another, for instance, when assigning strings to different variables.

~ Syntax:- char* strcpy(char* dest, const char* src);

- <u>strchr() Find Character in String:-</u> The strchr() function finds the first occurrence of a character in a string. It returns a pointer to the character's location or NULL if the character is not found. It is useful for searching for a character within a string, such as finding delimiters (like commas) or identifying specific characters.

~ Syntax:- char *strchr(const char *str, int c);

➢ **Example:-**

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

```
int main()
{    // Example variables
char str1[50] = "Hello";
char str2[] = "World!";
char str3[50];
char str4[50] = "HelloWorld!";
char str5[] = "Programming in C";     // strlen() - Get length of a string
printf("Length of str1: %zu\n", strlen(str1));     // strcpy() - Copy string
strcpy(str3, str1);
printf("Copied str1 into str3: %s\n", str3);     // strcat() - Concatenate strings
strcat(str1, str2);
printf("Concatenated str1 and str2: %s\n", str1);
// strncat() - Concatenate first n characters of str2 to str4
strncat(str4, str2, 3);  // Appends first 3 characters of str2 to str4
printf("Concatenated first 3 chars of str2 to str4: %s\n", str4);
// strcmp() - Compare two strings
if (strcmp(str1, str4) == 0)
{
printf("str1 and str4 are equal.\n");
}
else
{
printf("str1 and str4 are not equal.\n");
}     // strchr() - Find first occurrence of a character in a string
char *position = strchr(str5, 'C');
if (position != NULL)
{
printf("Character 'C' found at position: %ld\n", position - str5);
}
else
{
printf("Character 'C' not found in str5.\n");
}
return 0;
}
```

Output:-

```
Length of str1: 5
Copied str1 into str3: Hello
Concatenated str1 and str2: HelloWorld!
Concatenated first 3 chars of str2 to str4: HelloWorld!Wor
str1 and str4 are not equal.
Character 'C' found at position: 15
```

12. Structures in C:- Explain the concept of structures in C. Describe how to declare, initialize, and access structure members.

➢ A **structure** is a user-defined data type that allows grouping variables of different data types under a single name. Structures help organize data by providing a way to define a more complex data type that can store multiple variables (called members) in one object, even if they have varying data types.

- Structure Declaration:- We have to declare structure in C before using it in our program. In structure declaration, we specify its member variables along with their datatype. We can use the struct keyword.

~ Syntax:- struct structure_name
```
{    data_type member_name1;
     data_type member_name1;
  ... };
```

- Structure Definition:- To use structure in our program, we have to define its instance. We can do that by creating variables of the structure type. We can define structure variables using two methods:

~ Syntax:- // Structure Variable Declaration with Structure Template
```
struct structure_name
{    data_type member_name1;
     data_type member_name1;
        ....   }
variable1, varaible2, ...;
// Structure Variable Declaration after Structure Template
struct structure_name variable1, variable2, .......;
```

- Initialization Structure Members:- Provides initial values to structure variables. After declaring a structure, you can create variables of that structure type and initialize them.

~ Syntax:- struct StructureName variableName = {value1, value2, ...};

- Access Structure Members:- We can access structure members by using the (.) dot operator.

~ Syntax:-
```
structure_name.member1;
strcuture_name.member2;
```

~ Example:-
```
#include <stdio.h>
struct Person
{
char name[50];
int age;
float height;
};
int main()
{ // Declare and initialize a structure variable
struct Person person1 = {"Alice", 25, 5.6};
// Access structure members
printf("Name: %s\n", person1.name);
printf("Age: %d\n", person1.age);
printf("Height: %.1f\n", person1.height);
// Modify structure members
person1.age = 26; // Update age
printf("Updated Age: %d\n", person1.age);
return 0;
}
```

~ Output:-

```
Name: Alice
Age: 25
Height: 5.6
Updated Age: 26
```

13. **File Handling in C :- Explain the importance of file handling in C. Discuss how to perform file operations like creating, opening, closing, reading, and writing files.**

➢ **File handling** allows you to store data permanently by reading from and writing to files. Unlike variables, which store data temporarily, files enable long-term data storage, which is useful for data persistence between program executions.

➢ **Importance of File Handling:-**

~ Reusability: The data stored in the file can be accessed, updated, and deleted anywhere and anytime providing high reusability.

~ Portability: Without losing any data, files can be transferred to another in the computer system. The risk of flawed coding is minimized with this feature.

~ Efficient: A large amount of input may be required for some programs. File handling allows you to easily access a part of a file using few instructions which saves a lot of time and reduces the chance of errors.

~ Storage Capacity: Files allow you to store a large amount of data without having to worry about storing everything simultaneously in a program.

➢ **File Operation:-** File operations refer to the different possible operations that we can perform on a file in C such as:

- Creating a new file [fopen()]:- Creating a file is done using the fopen() function. When you open a file with write (w), append (a), or their binary equivalents (wb, ab), the file is created if it doesn't already exist. If it exists, it may be overwritten (with w) or opened in append mode (with a).
- Opening a File [fopen()]:- To open a file, use the fopen() function. It takes two arguments: the file name and the mode (e.g., "r" for reading, "w" for writing). If successful, fopen() returns a pointer to a FILE object representing the file. If the file cannot be opened, fopen() returns NULL.
- Reading from a File [fscanf() or fgets()]:- Reading from a file is done using functions like fscanf() for formatted text data, fgets() for reading strings, and fread() for binary data. These functions take the FILE pointer and read the file's contents based on the specified format and data type.
- Writing to a File [fprintf() or fputs()]:- To write to a file, use functions like fprintf() for formatted text output, fputs() for strings, and fwrite() for binary data. These functions take the FILE pointer and the data to be written, storing it in the specified format within the file.
- Moving to a specific location in a file [fseek(), rewind()]:- To move the file pointer within a file, use the fseek() function. This function takes three parameters: the FILE pointer, the offset, and the position from where to start (SEEK_SET, SEEK_CUR, or SEEK_END). You can use fseek() to navigate to a specific position in the file to read or write at that location.
- Closing a File [fclose()]:- The fclose() function is used to close an open file. This operation ensures that any data buffered in memory is written to the file before closing, freeing resources associated with the file. It takes the FILE pointer as an argument and returns zero on success, or EOF if an error occurs.