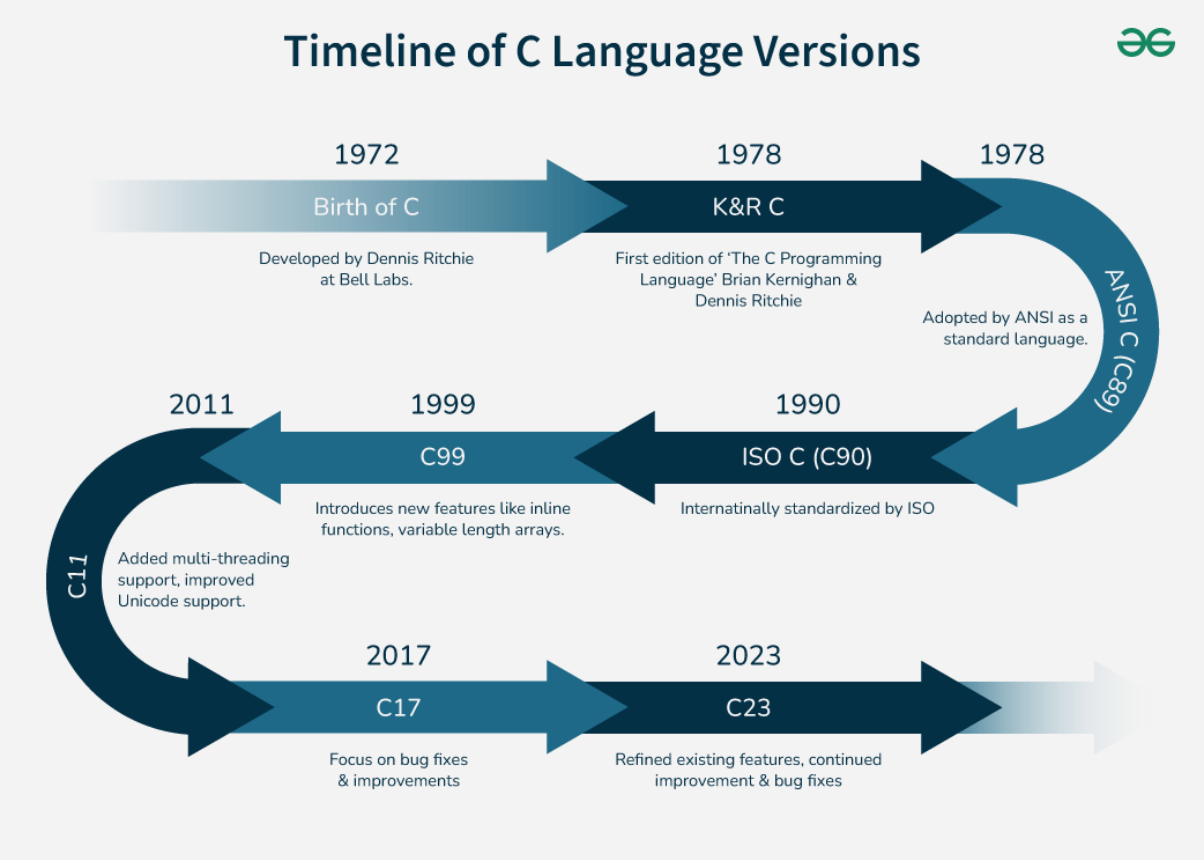
1. **Write an essay covering the history and evolution of C programming. Explain its importance and why it is still used today.**

C is a procedural general-purpose programming language that was originally created by Dennis Ritchie in 1972 in Bell Laboratories of AT&T Labs. It was originally intended as a system programming language to implement the UNIX system. C has become one of the world's most influential programming languages that is universally applied in a variety of applications owing to being memory-efficient, being able to provide low-level control, and being portable.

Standardization and Evolution

As C gained popularity, variations emerged across different compilers and platforms, necessitating standardization.



2.Research and provide three real-world applications where C programming is extensively used, such as in embedded systems, operating systems, or game development.

Building the Core: Operating Systems

Imagine a world without operating systems; phones or computers would be non-functioning shells. C takes center stage to create such basic cores.

The legendary Unix system, developed by Dennis Ritchie and Ken Thompson, was written almost entirely in C. That legacy lives on today. Major portions of Windows, Linux, and macOS rely on C because it allows programmers to interact directly with hardware, manage memory manually, and squeeze out every drop of performance.

Without C, the speed and dependability we've grown to expect from operating systems would be nothing more than a pipe dream.

Powering Embedded Systems

Look around: cars, microwave ovens, or smart thermostat likely has an embedded system inside. These small computers, buried in everyday appliances, depend heavily on C.

It's lean and provides programmers with precise control of scarce resources such as memory and processing power.

In a car's engine control unit, for instance, C ensures millisecond-level choices that keep you safe on the road. From medical devices to household appliances, C's ability to operate close to metal means it is invaluable in this type of work.

Crafting Compilers and Interpreters

Ever wonder how programming languages talk to machines? Compilers and interpreters bridge the gap, and a lot of them are written in C. The portability and speed of the language make it an ideal candidate to translate code, like Python or Java, into computer-understandable instructions. Early C++ compilers, for instance, were implemented in C. Even now, applications such as GCC (GNU Compiler Collection) rely on C to provide fast, consistent performance. It's a behind-the-scenes superhero that keeps the world of coding going.

Driving Game Development

Video games amaze us with breathtaking graphics and glitch-free action, and C has a part in that wonder. Though C++ tends to hog the limelight in today's game engines, C paved the way and continues to excel at certain things.

Veteran games such as Doom were developed in C and demonstrated its prowess for real-time performance. In the present age, it finds application in engines such as Unity (for lower-level operations) and in heavily resource-consuming simulations where milliseconds must be saved. Its ability to optimize speed remains in the arsenal of the gamer.

Enabling Networking Tools

The internet hums with activity, and C helps keep it running smoothly. Networking tools like routers, switches, and protocols owe a lot to this language.

Applications like Wireshark, which sniff network traffic, are constructed using C to process data packets effectively. Even components of the TCP/IP stack, the internet communication

backbone, have their origins in C. Its memory management and data processing capabilities ensure our networked world doesn't miss a beat.

Supporting Databases and Software

Behind each web search engine or internet shopping mall lies a database, and C powers the engines that propel them. MySQL and SQLite are programs that utilize C for its fundamental functions, giving speed and reliability.

Outside of databases, programs such as Adobe Photoshop and AutoCAD have relied on C for performance-critical parts. When you require software that's speedy and won't collapse under the strain, C comes to the rescue.

Why C Stays Relevant

So why does C survive when more modern languages vow simpler syntax or inherent functionality?

It's a matter of control and efficiency. C provides programmers memory access, a two-edged sword that is difficult to handle but yields optimum performance.

It's also transportable, it's a program on one machine, and it will likely work on another with minimal tweaking. That flexibility, combined with its minimalist design, makes C a go-to in fields where accuracy and speed matter more than convenience.

Final Thoughts

From the depths of operating systems to your smartwatch's circuits, C's fingerprints can be found. Its applications in the real world demonstrate a language that is at once a relic and a behemoth, affirming that great design is timeless.

Whether coder or curious onlooker, C's tale is a testament to how a humble tool can change the world, one line of code at a time.

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.

   Install a C Compiler:

- The most popular C compiler for Windows is MinGW. Download the MinGW installer and follow the installation instructions.
- During installation, make sure to select the option to install the “C Compiler.”

Install a Text Editor or IDE:

- You can use any text editor like Notepad, Visual Studio Code, or an IDE like Code::Blocks or Dev-C++ for a more feature-rich experience.
- Write your C code using the text editor or IDE.
- Open a command prompt, navigate to your code’s directory, and compile it using the gcc command. For example:
  gcc your_program.c -o your_program

Run Your C Program:

- After successful compilation, you can run your C program from the command prompt:
  your_program

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.

The basic structure of a C program is divided into 6 parts which makes it easy to read, modify, document, and understand in a particular format.
There are 6 basic sections responsible for the proper execution of a program.
Sections are mentioned below:

1. Documentation
2. Preprocessor Section
3. Definition
4. Global Declaration
5. Main() Function
6. Sub Programs

1. Documentation

This section consists of the description of the program, the name of the program, and the creation date and time of the program. It is specified at the start of the program in the form of comments. Documentation can be represented as:
// description, name of the program, programmer name, date, time etc.
or
/*
description, name of the program, programmer name, date, time etc.
*/
Anything written as comments will be treated as documentation of the program and this will not interfere with the given code. Basically, it gives an overview to the reader of the program.

2. Preprocessor Section

All the header files of the program will be declared in the preprocessor section of the program. Header files help us to access other's improved code into our code. A copy of these multiple files is inserted into our program before the process of compilation.

Example:

#include<stdio.h>
#include<math.h>

3. Definition

Preprocessors are the programs that process our source code before the process of compilation. There are multiple steps which are involved in the writing and execution of the program. Preprocessor directives start with the '#' symbol. The #define preprocessor is used to create a constant throughout the program. Whenever this name is encountered by the compiler, it is replaced by the actual piece of defined code.

Example:

```
#define long long ll
```

4. 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;
```

5. 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. void() main tells the compiler that the program will not return any value. The int main() tells the compiler that the program will return an integer value.

Example:

```
void main()
```

or

```
int main()
```

6. Sub Programs

User-defined functions are called in this section of the program. The control of the program is shifted to the called function whenever they are called from the main or outside the main() function. These are specified as per the requirements of the programmer.

Example:

```
int sum(int x, int y)
{
   return x+y;
}
```

Structure of C Program with example

Example: Below C program to find the sum of 2 numbers:

```
// Documentation
/**
 * file: sum.c
 * author: you
 * description: program to find sum.
 */
```

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

// Definition
#define X 20

// Global Declaration
int sum(int y);

// Main() Function
int main(void)
{
  int y = 55;
  printf("Sum: %d", sum(y));
  return 0;
}

// Subprogram
int sum(int y)
{
  return y + X;
}
```

Output

Sum: 75

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

Operators are the basic components of C programming. They are symbols that represent some kind of operation, such as mathematical, relational, bitwise, conditional, or logical computations, which are to be performed on values or variables. The values and variables used with operators are called **operands**.

Unary, Binary and Ternary Operators

On the basis of the number of operands they work on, operators can be classified into three types :

1. **Unary Operators:** Operators that work on single operand.
   Example: Increment(++) , Decrement(--)

2. **Binary Operators:** Operators that work on two operands.
   Example: Addition (+), Subtraction(-) , Multiplication (*)
3. **Ternary Operators:** Operators that work on three operands.
   Example: Conditional Operator(? :)

Types of Operators in C

C language provides a wide range of built in operators that can be classified into 6 types based on their functionality:

1. **Arithmetic Operators**
2. **Relational Operators**
3. **Logical Operator**
4. **Bitwise Operators**
5. **Assignment Operators**
6. **Other Operators**

Arithmetic Operators

The arithmetic operators are used to perform arithmetic/mathematical operations on operands.

Relational Operators:

In C, relational operators are the symbols that are used for comparison between two values to understand the type of relationship a pair of numbers shares. The result that we get after the relational operation is a boolean value, that tells whether the comparison is true or false. Relational operators are mainly used in conditional statements and loops to check the conditions in C programming.

Relational Operators in C

Operator Symbol	Operator Name
==	Equal To
!=	Not Equal To
<	Less Than
>	Greater Than
<=	Less Than Equal To
>=	Greater Than Equal To

Logical Operator

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**

Bitwise Operators

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.

Note: 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	**a | b**

Symbol	Operator	Description	Syntax
		and returns the result.	
^	**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 bits to the left by a given number of positions; multiplies the number by 2 for each shift.	**a << b**
>>	**Bitwise Rightshift**	Shifts bits to the right by a given number of positions; divides the number by 2 for each shift.	**a >> b**

Assignment Operators

Assignment operators are used to assign value to a variable. The left side operand of the assignment operator is a variable and the right side operand of the assignment

operator is a value. The value on the right side must be of the same data type as the variable on the left side otherwise the compiler will raise an error.
The assignment operators can be combined with some other operators in C to provide multiple operations using single operator. These operators are called compound operators.

Symbol	Operator	Description	Syntax
=	**Simple Assignment**	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	**a *= b**

Symbol	Operator	Description	Syntax
		to the left operand.	
/=	**Divide and assign**	Divide the left operand with the right operand and assign this value to the left operand.	**a /= b**
%=	**Modulus 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	**a |= b**

Symbol	Operator	Description	Syntax
		to the left operand.	
^=	**XOR and assign**	Performs bitwise XOR and assigns this value to the left operand.	**a ^= b**
>>=	**Rightshift and assign**	Performs bitwise Rightshift and assign this value to the left operand.	**a >>= b**
<<=	**Leftshift and assign**	Performs bitwise Leftshift and assign this value to the left operand.	**a <<= b**

Other Operators

Apart from the above operators, there are some other operators available in C used to perform some specific tasks. Some of them are discussed here:

sizeof Operator

sizeof() operator is a very useful tool that helps programmers understand how much memory a variable or data type occupies in the computer's memory.

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

```
    printf("Size of char: %lu bytes\n", sizeof(char));
    printf("Size of int: %lu bytes\n", sizeof(int));
    printf("Size of float: %lu bytes\n", sizeof(float));
    printf("Size of double: %lu bytes\n", sizeof(double));
    printf("Size of pointer: %lu bytes\n", sizeof(void *));
    return 0;
}
```

Output

Size of char: 1 bytes
Size of int: 4 bytes
Size of float: 4 bytes
Size of double: 8 bytes
Size of pointer: 8 bytes

Conditional or Ternary Operator (?:)

The **conditional operator in C** is kind of similar to the if-else statement as it follows the same algorithm as of if-else statement but the conditional operator takes less space and helps to write the if-else statements in the shortest way possible. It is also known as the **ternary operator in C** as it operates on three operands.

Syntax of Conditional/Ternary Operator in C

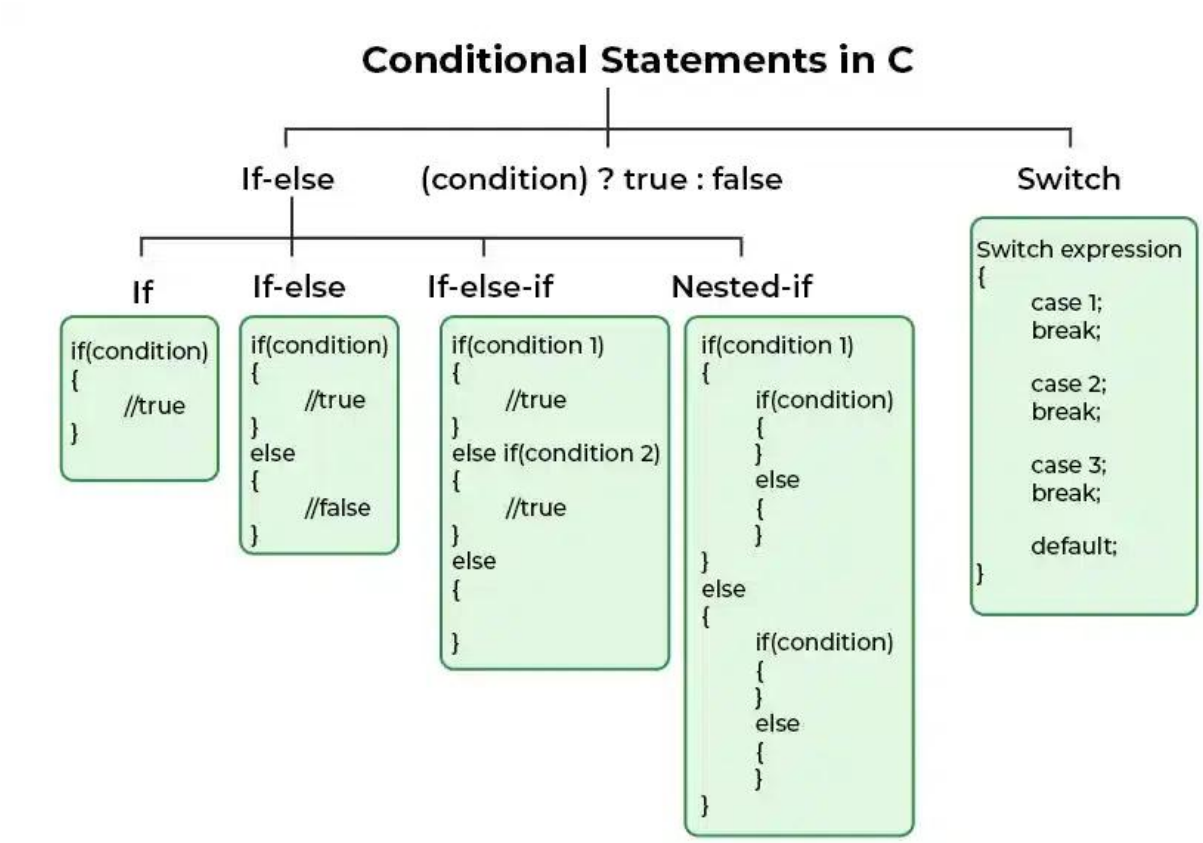
The conditional operator can be in the form

variable = Expression1 **?** *Expression2* **:** *Expression3;*

5.Control Flow Statements in C

Explain decision-making statements in C (if, else, nested if-else, switch). Provide examples of each.

In C, programs can choose which part of the code to execute based on some condition. This ability is called **decision making** and the statements used for it are called **conditional statements.** These statements evaluate one or more conditions and make the decision whether to execute a block of code or not.



1. if in C

The if statement is the simplest decision-making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statements is executed otherwise not.
A **condition** is any expression that evaluates to either a true or false (or values convertible to true or flase).

2. if-else in C

The **if** statement alone tells us that if a condition is true, it will execute a block of statements and if the condition is false, it won't. But what if we want to do something else when the condition is false? Here comes the C **else** statement. We can use the **else** statement with the **if** statement to execute a block of code when the condition is false. The if-else statement consists of two blocks, one for false expression and one for true expression.

3. Nested if-else in C

A nested if in C is an if statement that is the target of another if statement. Nested if statements mean an if statement inside another if statement. Yes, C allow us to nested if statements within if statements, i.e, we can place an if statement inside another if statement.

4. if-else-if Ladder in C

The if else if statements are used when the user has to decide among multiple options. The C if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed,

and the rest of the C else-if ladder is bypassed. If none of the conditions is true, then the final else statement will be executed. if-else-if ladder is similar to the switch statement.

5. switch Statement in C

The switch case statement is an alternative to the if else if ladder that can be used to execute the conditional code based on the value of the variable specified in the switch statement. The switch block consists of cases to be executed based on the value of the switch variable.

6. Conditional Operator in C

The conditional operator is used to add conditional code in our program. It is similar to the if-else statement. It is also known as the ternary operator as it works on three operands.

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 repeat statements:

➢ For- Used to efficiently write a loop that needs to execute a specific number of times ,fixed loops.

➢ While- It repeatedly executes a target statement as long as the given condition is true ,condition-based.

➢ Do-while- Do-while loop is similar to while loop , except the fact that it executes once even if condition is false ,runs at least once.

7.Loop Control Statements

Explain the use of break, continue, and goto statements in C. Provide examples of each.

→ Break- used inside loop or switch statement

→ Continue- also inside a loop, compiler will skip all the following statements in the loop and resume the next loop iteration.

→ Goto- we can transfer the control from current location to anywhere in the program.

8.Functions in C

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

→ A function in C is a block of code that performs a specific task. Functions are reusable blocks of code. Steps: ➢ Declaration- Tells the compiler the function's name, return type, and parameters. Placed before main() or in a header file. ➢ Definition- Contains the actual code (body) of the function. Specifies what the function does. ➢ Calling- Executes the function. Passes arguments (if any) to the function.

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 data structure that stores multiple values of the same data type in contiguous memory locations. → One-Dimensional Array: It consists of a single row of elements, accessible using a single index. ➢ For example, → int arr; stores five integers in a row. → Two-Dimensional Array: It consists of a grid of elements, accessible using two indices. ➢ For example, → int matrix = {{10, 20}, {30, 40}}; stores a 2x3 matrix.

10. Pointers in C

• THEORY EXERCISE: o Explain what pointers are in C and how they are declared and initialized. Why are pointers important in C?

Pointers in C are variables that store the memory address of another variable.

→ They are declared using the syntax data_type *pointer_name, where the asterisk (*) indicates the pointer.

→ They are essential for creating data structures like linked lists and trees, and for passing variables by value to functions.

11.Strings in C

Explain string handling functions like strlen(), strcpy(), strcat(), strcmp(), and strchr(). Provide examples of when these functions are useful.

strlen(): Returns the length of a string (excluding the null terminator). Example: char str[] = "Hello, World!"; int length = strlen(str);.

→ strcpy(): Copies the contents of one string to another. Example: char source[] = "Hello"; char destination[] = "World"; strcpy(destination, source);.

→ strcat(): Concatenates two strings and appends the result to the destination string. Example: char destination; strcat(destination, "World");.

→ strcmp(): Compares two strings lexicographically and returns an integer. Example: int comparison = strcmp("Hello", "World");.

→ strncpy(): Copies at most n characters from the source string to the destination string. Example: char destination; strncpy(destination, source, 3);.

→ strncat(): Similar to strcat(), but only appends up to n characters from the source string. Example: char destination; strncat(destination, source, 3);.

→ These functions are useful for various tasks such as string manipulation, comparison, and concatenation in C programming.

12. Structures in C

• Explain the concept of structures in C. Describe how to declare, initialize, and access structure members.

→ In C, a structure is a user-defined data type that allows grouping different types of data under a single name.

→ For example:

➢ struct Student {
char name; [^1^]
float marks;
};
Student student;

→ To declare a structure, use the struct keyword followed by the structure name and its members.

→ For instance:

➢ student.name = "Amit"; [^1^]
student.marks = 92.5;.

→ also initialize structure members using an initializer list:

➢ Student student = { "Amit", 92.5 };

→ To initialize a structure, you can use the dot operator (.) or the arrow operator (->).
→ To access structure members, use the dot operator
➢ printf("Name: %s, Marks: %.2f
", student.name, student.marks);

13. File Handling in C.

• Explain the importance of file handling in C. Discuss how to perform file operations like opaening, closing, reading, and writing files.

→ File handling in C is essential for storing and retrieving data permanently. It allows programs to create, open, read, write, and close files, enabling efficient data management.
→ fopen(): Opens a file in read, write, or append mode. It checks for the file's existence and creates it if it doesn't exist.
→ fclose(): Closes a file after reading or writing operations are completed.
→ fread(): Reads data from a file into a specified buffer.
→ fwrite(): Writes data from a specified buffer to a file.
→ fscanf(): Reads data from a file into variables.
→ fprintf(): Writes data from a specified buffer to a file.
→ File handling is crucial for applications that require persistent data storage, such as logging, configuration files, and user data.