**Q1. Write an essay covering the history and evolution of C programming. Explain its importance and why it is still used today.**

* The History and Importance of C Programming

C programming is one of the most important languages in computer science. It was developed in the early 1970s at Bell Laboratories by **Dennis Ritchie**, evolving from the B language created by Ken Thompson. C was originally designed to improve system programming and was used to develop the **UNIX operating system**, which made both UNIX and C widely popular.

The importance of C lies in its unique qualities:

**Efficiency and performance** for low-level system tasks.

**Portability**, allowing programs to run across different machines.

**Educational value**, teaching fundamental concepts like memory management and data structures.

**Flexibility**, making it suitable for both small applications and large systems.

Even today, C is widely used in:

**Operating systems** (Linux, UNIX, Windows components).

**Embedded systems** (microcontrollers and IoT devices).

**Performance-critical applications** (databases, compilers, real-time software).

**Maintaining legacy code**, as vast systems are still written in C.

**Q2 . Explain the basic structure of a C program, including headers, main function, comments, data types, and variables. Provide examples.**

**Basic Structure of a C Program**

A C program follows a specific structure to ensure clarity, readability, and proper execution. Below are the key components of a C program, explained with examples:

**1. Documentation Section (Comments)**

* Purpose: Used to describe the program, its purpose, or any additional notes for developers.
* Syntax:
  + Single-line comment: //
  + Multi-line comment: /\* ... \*/

**2. Preprocessor Directives (Header Files)**

* **Purpose**: Include standard or user-defined libraries for specific functionalities.
* **Syntax**: #include <header\_file>

**Example**:

#include <stdio.h> // Standard Input/Output library

#include <math.h> // Math functions library

**3. main() Function**

* **Purpose**: The entry point of the program where execution begins.

int main() {

// Code

return 0;

}

**Q3. Write notes explaining each type of operator in C: arithmetic, relational, logical, assignment, increment/decrement, bitwise, and conditional operators.**

1. Arithmetic Operators

* Purpose: Perform basic mathematical operations.
* Operators: + (addition), - (subtraction), \* (multiplication), / (division), % (modulus - remainder after division)
* Note: Increment/Decrement Operators: ++ increases by 1, -- decreases by 1.

int x = 5;

x++; // x becomes 6

x--; // x becomes 5

2. **Relational Operators**

* Purpose: Compare two values, return true (1) or false (0).
* Operators: == (equal), != (not equal), > (greater than), < (less than), >= (greater or equal), <= (less or equal)

5 == 3; // 0 (false)

4 != 4; // 0 (false)

9 > 7; // 1 (true)

2 <= 2; // 1 (true)

3**. Logical Operators**

* Purpose: Combine or invert conditions.
* Operators: && (logical AND), || (logical OR), ! (logical NOT)

(a > 5 && b < 10); // true if both conditions are true

(x == 0 || y != 0); // true if either condition is true

!(a == b); // true if a is not equal to b

**4. Assignment Operators**

* Purpose: Assign values to variables, sometimes combined with arithmetic.
* Main operator: =
* Shorthand operators: +=, -=, \*=, /=, %=
* Examples:

**int** x = 5;

x += 3; *// x = x + 3, x becomes 8*

x \*= 2; *// x = x \* 2, x becomes 16*

**5. Increment/Decrement Operators (Unary Operators)**

* Purpose: Increase or decrease a variable by 1.
* Operators: ++ (increment), -- (decrement)
* Forms: Prefix (++x) or Postfix (x++)
* Example:

**int** x = 5;

++x; *// increments then returns: x becomes 6*

x++; *// returns then increments: x still 6 but becomes 7 after*

**6. Bitwise Operators**

* Purpose: Perform operations on bits of numbers.
* Operators:
  + & (AND)
  + | (OR)
  + ^ (XOR)
  + ~ (NOT)
  + << (Left shift)
  + >> (Right shift)

Example :

int a = 5; // binary 0101

int b = 3; // binary 0011

int c = a & b; // binary 0001 = 1

int d = a << 1; // binary 1010 = 10

**7. Conditional (Ternary) Operator**

* Purpose: Simplify simple if-else statements into a single line.
* Syntax: condition ? expression\_if\_true : expression\_if\_false;
* Example:

**int** x = 10, y;

y = (x > 5) ? 100 : 200; *// y = 100 because condition is true*

**Q4. Explain decision-making statements in C (if, else, nested if-else, switch). Provide examples of each.**

**Decision-Making Statements in C** are used to control the flow of a program based on certain conditions. These statements allow the program to execute specific blocks of code depending on whether a condition evaluates to true or false. Below are the key decision-making statements in C, along with examples:

**1. if Statement**

The if statement executes a block of code only if the condition is **true**.

**Syntax:**

if (condition) {

// code to execute if condition is true

}

**Example:**

#include <stdio.h>

int main() {

int age = 20;

if (age >= 18) {

printf("You are eligible to vote.\n");

}

return 0;

}

**2. if-else Statement**

Executes one block if the condition is **true**, otherwise another block.

**Syntax:**

if (condition) {

// code if true

} else {

// code if false

}

**Example:**

#include <stdio.h>

int main() {

int number = 15;

if (number % 2 == 0) {

printf("The number is Even.\n");

} else {

printf("The number is Odd.\n");

}

return 0;

}

**3. Nested if-else Statement**

An if-else statement **inside another if-else**.  
Used when we have multiple conditions.

**Syntax:**

if (condition1) {

// code if condition1 is true

} else {

if (condition2) {

// code if condition2 is true

} else {

// code if all are false

}

}

**Example:**

#include <stdio.h>

int main() {

int marks = 75;

if (marks >= 90) {

printf("Grade: A\n");

} else if (marks >= 75) {

printf("Grade: B\n");

} else if (marks >= 50) {

printf("Grade: C\n");

} else {

printf("Grade: Fail\n");

}

return 0;

}

**4. switch Statement**

The switch statement allows multi-way branching.  
It’s often used instead of multiple if-else when checking for equality.

**Syntax:**

switch (expression) {

case value1:

// code

break;

case value2:

// code

break;

...

default:

// code if no case matches

}

**Example:**

#include <stdio.h>

int main() {

int day = 3;

switch (day) {

case 1:

printf("Monday\n");

break;

case 2:

printf("Tuesday\n");

break;

case 3:

printf("Wednesday\n");

break;

case 4:

printf("Thursday\n");

break;

case 5:

printf("Friday\n");

break;

case 6:

printf("Saturday\n");

break;

case 7:

printf("Sunday\n");

break;

default:

printf("Invalid day number!\n");

}

return 0;

}

**Summary:**

* if → single condition.
* if-else → condition with an alternative.
* nested if-else → multiple conditions.
* switch → multiple fixed choices (best for equality checking).

Q5.