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

History and Evolution of C Programming:
C was developed in 1972 by Dennis Ritchie at Bell Laboratories. It was created to develop the UNIX operating system and was derived from the B language, which in turn came from BCPL. The primary goal was to create a structured and efficient programming language that offered low-level memory access and could replace assembly language in systems programming.

Over the years, C has undergone several standardizations:

- 1978: First edition of "The C Programming Language" by Kernighan and Ritchie.
- 1989: ANSI C (C89) standardized by the American National Standards Institute.
- 1999: C99 introduced features like inline functions and new data types.
- 2011: C11 added multi-threading support and better Unicode handling.
- 2017 & 2023: C17 and C23 refined the language with bug fixes and minor improvements.

Importance and Continued Use:

- C is known for its performance and efficiency.
- It provides low-level access to memory.
- It is portable across various platforms.
- It forms the basis for many other languages like C++, Java, and Python.
- Widely used in embedded systems, operating systems, and system-level programming.

---

2. 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.

Installing GCC (via MinGW on Windows):

1. Download MinGW from the official website.
2. Run the installer and select "gcc-g++", "binutils", and "mingw32-base".
3. Add the path to MinGW's bin folder (e.g., C:\MinGW\bin) to the system PATH environment variable.

Setting Up IDEs:

- **DevC++:**
  1. Download and install DevC++.
  2. Create a new project or source file.
  3. Write and compile C code.
- **VS Code:**
  1. Install Visual Studio Code.

    2. Install the "C/C++" extension by Microsoft.
    3. Set up tasks.json and launch.json for build and debug configuration.
- **CodeBlocks:**
    1. Download the version that includes the compiler.
    2. Install and open CodeBlocks.
    3. Create a new project and write C code.

---

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

Basic Structure Example:

```
#include <stdio.h> // Header file

int main() {
   // Single-line comment
   int number = 10; // Variable declaration
   printf("Number is %d", number);
   return 0;
}
```

Key Elements:

- **Headers:** `#include <stdio.h>` includes standard input-output functions.
- **Main Function:** `int main()` is the entry point of the program.
- **Comments:** Used to explain code (`//` and `/* */`).
- **Data Types:** int, float, char, etc.
- **Variables:** Store data values.

---

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

Operators in C - Theory and Notes

In C programming, **operators** are special symbols used to perform operations on variables and values. These operations can be arithmetic, logical, comparison, bitwise manipulation, etc.

C supports the following types of operators:

🔢 1. Arithmetic Operators

Purpose: Perform basic mathematical operations on numeric values.

Operator	Description	Example	Result (a = 10, b = 3)
+	Addition	a + b	13
-	Subtraction	a - b	7
*	Multiplication	a * b	30
/	Division	a / b	3
%	Modulus	a % b	1

□ **Example:**

```
c
CopyEdit
int a = 10, b = 3;
printf("Sum = %d", a + b); // Output: 13
```

🔍 2. Relational Operators

Purpose: Compare two values or expressions and return a boolean result (0 or 1).

Operator	Meaning	Example	Result (a = 5, b = 10)
==	Equal to	a == b	0 (false)
!=	Not equal to	a != b	1 (true)
>	Greater than	a > b	0
<	Less than	a < b	1
>=	Greater or equal	a >= b	0
<=	Less or equal	a <= b	1

□ **Example:**

```
c
CopyEdit
if (a < b) {
   printf("a is less than b");
}
```

⚙ 3. Logical Operators

Purpose: Combine multiple conditions or expressions logically.

Operator	Name	Description
&&	Logical AND	True if both conditions are true
`		`
!	Logical NOT	Reverses the truth value of the condition

□ Example:

c
CopyEdit

```
if (a > 0 && b > 0) {
   printf("Both numbers are positive");
}
```

□ 4. Assignment Operators

Purpose: Assign values to variables.

Operator	Description	Example	Equivalent To
=	Simple assignment	a = b	assign b to a
+=	Add and assign	a += b	a = a + b
-=	Subtract and assign	a -= b	a = a - b
*=	Multiply and assign	a *= b	a = a * b
/=	Divide and assign	a /= b	a = a / b
%=	Modulus and assign	a %= b	a = a % b

□ Example:

c
CopyEdit

```
int a = 10;
a += 5;  // a = a + 5 → a becomes 15
```

🔁 5. Increment and Decrement Operators

Purpose: Increase or decrease a variable's value by 1.

Operator	Type	Example	Effect
++	Increment	++a	Pre-increment

Operator	Type	Example	Effect
a++	Increment	a++	Post-increment
--	Decrement	--a	Pre-decrement
a--	Decrement	a--	Post-decrement

□ Example:

```
c
CopyEdit
int a = 5;
printf("%d", ++a); // Output: 6
```

🔧 6. Bitwise Operators

Purpose: Perform operations at the binary level. Mostly used in systems programming.

Operator	Name	Example	Description
&	AND	a & b	Bitwise AND
`	`	OR	`a
^	XOR	a ^ b	Bitwise Exclusive OR
~	NOT	~a	Bitwise complement
<<	Left shift	a << 1	Shift bits to the left
>>	Right shift	a >> 1	Shift bits to the right

□ Example:

```
c
CopyEdit
int a = 5, b = 3;
printf("%d", a & b); // Output: 1
```

❓ 7. Conditional (Ternary) Operator

Purpose: Short form of an if-else statement. Evaluates a condition and returns a value based on the result.

Syntax:

```
c
CopyEdit
```

```
condition ? value_if_true : value_if_false;
```

□ Example:

c
CopyEdit

```
int a = 10, b = 20;
int max = (a > b) ? a : b;
printf("Max = %d", max);  // Output: 20
```

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

- **if statement:**

```
if (x > 0) {
   printf("Positive");
}
```

- **if-else statement:**

```
if (x % 2 == 0) {
   printf("Even");
} else {
   printf("Odd");
}
```

- **nested if-else:**

```
if (x > 0) {
   if (x < 100) {
      printf("Positive and less than 100");
   }
}
```

- **switch statement:**

```
switch (choice) {
   case 1: printf("Option 1"); break;
   case 2: printf("Option 2"); break;
   default: printf("Invalid");
}
```

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

Loop Type	Use Case	Syntax Example
while	When the number of iterations is unknown	while(condition)
for	When iterations are fixed or count-based	for(i=0; i<10; i++)

Loop Type	Use Case	Syntax Example
do-while	At least one iteration is required	do { } while(condition);

7. Explain the use of break, continue, and goto statements in C.

- **break:** Exits the loop prematurely

```
for (int i=0; i<5; i++) {
    if (i == 3) break;
    printf("%d ", i);
}
```

- **continue:** Skips current iteration

```
for (int i=0; i<5; i++) {
    if (i == 2) continue;
    printf("%d ", i);
}
```

- **goto:** Jumps to a labeled section

```
goto label;
printf("Skipped\n");
label:
printf("Jumped here\n");
```

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

- **Declaration:**

```
int add(int, int);
```

- **Definition:**

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

- **Function Call:**

```
int result = add(5, 3);
```

9. Explain the concept of arrays in C. Differentiate between one-dimensional and multi-dimensional arrays.

- **One-Dimensional Array:**

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

- **Multi-Dimensional Array:**

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

Arrays store multiple elements of the same data type in contiguous memory locations.

---

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

Pointers: Variables that store memory addresses.

```
int a = 10;
int *ptr = &a;
```

Importance:

- Dynamic memory management
- Efficient array and structure handling
- Function argument passing (call by reference)

---

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

- strcpy(dest, src) – Copies one string to another
- strcat(dest, src) – Concatenates strings
- strcmp(s1, s2) – Compares two strings
- strchr(str, ch) – Finds a character in string

Example:

```
char s1[20] = "Hello";
char s2[20];
strcpy(s2, s1);
printf("%s", s2);
```

---

12. Explain the concept of structures in C. Describe how to declare, initialize, and access structure members.Structures: User-defined data types to group different data types.

```
struct Student {
   int id;
   char name[20];
};

struct Student s1 = {1, "John"};
```

```
printf("%d %s", s1.id, s1.name);
```

---

13. Explain the importance of file handling in C. Discuss how to perform file operations like opening, closing, reading, and writing filesImportance: File handling allows programs to store data permanently.

Operations:

```
FILE *fptr;
fptr = fopen("data.txt", "w");
fprintf(fptr, "Hello");
fclose(fptr);
```

- fopen() – Opens file
- fprintf()/fscanf() – Writes/reads formatted data
- fclose() – Closes file