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 programming was developed in the early 1970s at Bell Labs by **Dennis Ritchie**. It evolved from the B language and was mainly designed to develop the **Unix Operating System**.

****Importance of C:****

- **Foundation of Modern Languages** – C influenced C++, Java, and Python.
- **Portability** – Programs can run on different platforms with minimal changes.
- **Efficiency** – Provides low-level memory access and direct hardware control.
- **Versatility** – Used in system programming, embedded systems, and compilers.

****Why Still Used Today:****

C remains popular because it is **fast, portable, and close to hardware**, making it ideal for **operating systems, embedded devices, and high-performance applications**.

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.

****Steps to Install GCC Compiler:****

1. **Windows** → Install **MinGW** or **TDM-GCC**.
2. **Linux** → Open terminal and run: `sudo apt-get install build-essential`
3. **MacOS** → Install **Xcode Command Line Tools**.

****Setting up an IDE:****

- **DevC++** → Simple, comes pre-configured with GCC.
- **VS Code** → Install the **C/C++ extension**, link GCC, configure tasks.json.
- **CodeBlocks** → Download version with MinGW; automatically configures GCC.

****Conclusion:****

A compiler + IDE makes it easier to **write, compile, debug, and run C programs**.

3. 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:****

1. **Headers** → Libraries used with `#include`.
2. **main() Function** → Starting point of execution.
3. **Comments** → Notes for programmers (`//` single-line, `/* */` multi-line).
4. **Data Types** → int, float, char, double, etc.
5. **Variables** → Named memory locations for storing values.

****Example Program:****

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

int main() {
   // This is a single-line comment
   int age = 20;        // Integer variable
   float salary = 25000; // Floating-point variable

   printf("Age: %d\n", age);
   printf("Salary: %.2f\n", salary);
   return 0;
}
```

****Explanation:****

- `#include <stdio.h>` → Imports standard input/output functions.
- `int main()` → Main entry function.
- `printf()` → Displays output on screen.

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

****Arithmetic Operators (+, -, *, /, %):**

Perform basic mathematical operations.

****Relational Operators (==, !=, >, <, >=, <=):**

Compare values.

****Logical Operators (&&, ||, !):**

Used to combine or invert conditions.

****Assignment Operators (=, +=, -=, *=, /=, %=):**

Assign values to variables.

****Increment/Decrement Operators (++ , --):**

Increase or decrease value by 1.

****Bitwise Operators (&, |, ^, ~, <<, >>):**

Work on bits directly.

****Conditional (?:):**

Ternary operator to replace simple if-else.

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

****if Statement:**** Executes a block if condition is true.
****else Statement:**** Executes if condition is false.
****Nested if-else:**** Multiple conditions.
****switch Statement:**** Selects one block from many options.

****Example:****
```c
int num = 3;
if(num > 0) {
   printf("Positive");
} else if(num < 0) {
   printf("Negative");
} else {
   printf("Zero");
}

switch(num) {
   case 1: printf("One"); break;
   case 2: printf("Two"); break;
   default: printf("Other");
}
```

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

****while Loop:**** Condition checked first, runs while true. Best when repetitions unknown.
****for Loop:**** Initialization, condition, increment in one line. Best when count is known.
****do-while Loop:**** Executes at least once, condition checked after loop. Best when loop must run once.

****Example:****
```c
int i = 0;
while(i < 5) { printf("%d", i); i++; }
for(int j=0;j<5;j++){ printf("%d", j); }
int k=0; do{ printf("%d", k); k++; }while(k<5);
```

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

break: Exits from loop/switch immediately.
continue: Skips current iteration and goes to next.
goto: Jumps to a labeled statement (not recommended for structured code).

Example:
```c
for(int i=0;i<5;i++){
   if(i==2) continue;
   if(i==4) break;
   printf("%d", i);
}

goto label;
printf("Skipped");
label: printf("Reached here");
```

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

Declaration: Tells compiler about function name, return type, parameters.
Definition: Provides body of function.
Call: Executes the function.

Example:
```c
int add(int a, int b); // Declaration

int main(){
   int sum = add(3,4); // Call
   printf("%d", sum);
   return 0;
}

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

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

****Array:**** Collection of elements of same type stored in contiguous memory.

****1D Array:**** Linear list of elements.
```c
int arr[5] = {1,2,3,4,5};
```

****2D Array (Matrix):**** Table of rows and columns.
```c
int mat[2][3] = {{1,2,3},{4,5,6}};
```

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

****Pointer:**** A variable that stores the memory address of another variable.

****Declaration:**** `int *p;`
****Initialization:**** `int a=10; int *p=&a;`

****Importance:****
- Direct memory access
- Dynamic memory allocation
- Useful in arrays, strings, and functions

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 string.
****strcat(s1, s2):**** Concatenates two strings.
****strcmp(s1, s2):**** Compares two strings.
****strchr(str, c):**** Finds first occurrence of char in string.

****Example:****
```c
char str1[20] = "Hello";
char str2[20] = "World";
printf("%d", strlen(str1));
strcpy(str2, str1);
strcat(str1, " Everyone");
int cmp = strcmp("a","b");
char *ptr = strchr(str1,'o');
```

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

****Structure:**** User-defined data type that groups different data types.

****Declaration:****
```c
struct Student {
   int id;
   char name[20];
   float marks;
};
```

****Initialization and Access:****
```c
struct Student s1 = {1,"John",85.5};
printf("%d %s %f", s1.id, s1.name, s1.marks);
```

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

Importance: File handling allows permanent storage of data.

Opening: `fopen("file.txt","w");`
Closing: `fclose(fp);`
Writing: `fprintf(fp,"Hello");`
Reading: `fscanf(fp,"%s",str);`

Example:
```c
FILE *fp;
fp = fopen("test.txt","w");
fprintf(fp,"Hello World");
fclose(fp);
```