modul 2

1 - (1) C Programming – Short Note

Developed by: Dennis Ritchie in 1972 at Bell Labs.

Purpose: To write the UNIX operating system.

Based on: B and BCPL languages.

Standardized: ANSI C (1989) – for uniformity.

(2) Importance of C Language

Fast and efficient

Portable across systems

Direct memory access

Strong base for learning programming

(3) Still Used Today For:

Operating systems (Linux, Windows)

Embedded systems (TV, remote, microwave)

Device drivers & system tools

Teaching programming basics

2 - (1) GCC Installation (Windows using MinGW)

Go to https://www.mingw-w64.org/

Download the installer for Windows.

Run the setup ? Choose architecture (x86_64) ? Install.

After install, add bin folder path to System Environment Variables > Path
(Example: C:\Program Files\mingw-w64\bin)

Open CMD ? type gcc --version to check if installed.

(2) Using Dev C++ (Easiest for Beginners)

Download from https://sourceforge.net/projects/orwelldevcpp/

Install DevC++ (it comes with GCC pre-installed).

Open DevC++ ? New File ? Source File ? Save with .c extension.

Write C code ? Press F9 to compile and run.

(3) Using Code::Blocks

Go to https://www.codeblocks.org/downloads/

Download version: with MinGW included.

Install and launch Code::Blocks.

Go to File ? New ? Project ? Console Application ? C.

Write code and press Build and Run.

3 - 1. Header Files

Use #include to add libraries.

Example: #include <stdio.h> for input/output functions.

2. Comments

Help explain the code.

Single-line: // this is a comment

Multi-line:

3. Main Function

Entry point of the program:

```
int main()
{
   // code
  return 0;
}
```

4. Data Types
int ,float,double,char

5. Variables

Used to store data.

Must be declared with a data type:

```
int age = 20;
float price = 99.50;
```

4 - ? Arithmetic Operators

+	Addition	a + b
-	Subtraction	a - b
*	Multiplication	a * b
/	Division	a / b
%	Modulus	a % b (remainder)

? Relational Operators

Used to compare values.

==	Equal to	a == b
!=	Not equal to	a != b
>	Greater than	a > b
<	Less than	a < b
>=	Greater or equal	a >= b
<=	Less or equal	a <= b

? Logical Operators

Used for logic-based decisions.

&&	AND	a > 5 && b < 10
`		`
!	NOT	!(a == b)

? Assignment Operators

Used to assign values to variables.

=	Assign value	a = 5
+=	Add and assign	a += 1
-=	Subtract and assign	a -= 1

*=	Multiply and assign	a *= 2
/=	Divide and assign	a /= 2

? Increment/Decrement Operators

++	Increment by 1	a++ or ++a
--	Decrement by 1	a-- or --a

? Bitwise Operators

Used for bit-level operations.

&	AND	a & b
`	`	OR
^	XOR	a ^ b
~	NOT	~a
<<	Left shift	a << 2
>>	Right shift	a >> 2

? Conditional (Ternary) Operator

```
int result = (a > b) ? a : b;
```

5 - ① if Statement

Used to execute a block only if condition is true.

```
int num = 10;
if (num > 0) {
printf("Positive number");
}
```

② if-else Statement

Used to choose between two blocks based on condition.

```
    int num = -5;
    if (num >= 0) {
printf("Positive");
    } else {
printf("Negative");
    }
```

③ nested if-else Statement

if inside another if — used for multiple conditions.

```
      int marks = 85;
      if (marks >= 90) {
printf("Grade A");
      } else {
if (marks >= 75) {
   printf("Grade B");
} else {
   printf("Grade C");
}
      }
```

④ switch Statement

Used when you have multiple fixed options (like menu choice).

```
      int day = 3;
      switch(day) {
case 1: printf("Monday"); break;
case 2: printf("Tuesday"); break;
case 3: printf("Wednesday"); break;
default: printf("Invalid day");
      }
```

6 - 1. for loop

```
for (int i = 1; i <= 5; i++) {
  printf("%d\n", i);
}
```

2. while loop

```
int i = 1;
while (i <= 5) {
 printf("%d\n", i);
 i++;
}
```

3. do-while loop

```
int i = 1;
do {
  printf("%d\n", i);
  i++;
} while (i <= 5);
```

7 - 1. break Statement

Used to exit a loop or switch early.
Example:

```
for (int i = 1; i <= 10; i++) {
  if (i == 5) {
    break;
 }
  printf("%d\n", i);
}
```

2. continue Statement

Skips the current iteration and moves to the next loop cycle.
Example:

```
for (int i = 1; i <= 5; i++) {
  if (i == 3) {
    continue;
}
 printf("%d\n", i);
}
```

3. goto Statement

Jumps to a labeled part of the code. (⚠️ Use carefully)
Example:

```
int i = 1;
start:
printf("%d\n", i);
i++;
if (i <= 3) {
   goto start;
}
```

8 - What are Functions in C?

A function is a block of code that performs a specific task.
It helps in reusability and keeps code modular and organized.

① Function Declaration

Tells the compiler about function name, return type, and parameters.

```
int add(int a, int b);
```

② Function Definition

Actual code block of the function.

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

③ Function Call

Use the function inside main() or another function.

```
int result = add(5, 3);
printf("Sum = %d", result);
```

Example :-

```
#include <stdio.h>

int add(int, int);

int main() {
  int sum = add(4, 6);
 printf("Sum = %d", sum);
 return 0;
}


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

9 - What is an Array in C?

An array is a collection of elements (same data type) stored in contiguous memory locations.

Helps in storing multiple values in a single variable.

1. One-Dimensional (1D) Array

Linear list of elements.
Syntax:

```
int numbers[5] = {10, 20, 30, 40, 50};
```

Example:

```
#include <stdio.h>
int main() {
   int marks[3] = {70, 80, 90};
  printf("Second mark: %d", marks[1]);
   return 0;
}
```

2. Multi-Dimensional Array (e.g., 2D)

Used to store data in rows and columns (like a matrix).
Syntax:

```
int matrix[2][3] = {
  {11, 12, 13},
  {42, 53, 69}
};
```

Example:

```
#include <stdio.h>
int main() {
int a[2][2] = {{1, 2}, {3, 4}};
printf("Value: %d", a[1][0]);  // prints 3
return 0;
}
```

10 - A pointer is a variable that stores the memory address of another variable.

Declaration of Pointer
```
int *ptr;
```

Initialization of Pointer
```
int x = 10;
int *ptr = &x;
```

Why Are Pointers Important in C?

Memory Access -   Allows direct access to memory using addresses.
Efficiency - Speeds up performance by avoiding data copying.
Dynamic Memory Allocation - Needed for malloc(), calloc() etc. in heap memory.
Array & String Handling   - Useful in pointer arithmetic and character arrays.
Function Arguments (Pass-by-Reference) - Allows modifying variables from called function.

11 - 1. strlen() – String Length

-> Returns the number of characters in a string (excluding \0).

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

        int main() {
        char str[] = "Hello";
        printf("Length = %lu", strlen(str));  // Output: 5
        return 0;
}
```

2. strcpy() – String Copy

-> Copies one string into another.

```
        char src[] = "C Language";
        char dest[20];
        strcpy(dest, src);  // dest = "C Language"
```

3. strcat() – String Concatenation

-> Appends one string at the end of another.

```
        char s1[20] = "Hello ";
        char s2[] = "World";
        strcat(s1, s2);
```

4. strcmp() – String Comparison

-> ompares two strings:

Returns 0 if equal

Negative if first < second

Positive if first > second

```
        char a[] = "apple";
        char b[] = "banana";
        int result = strcmp(a, b);
```

12 - What is a Structure in C?

A structure is a user-defined data type that groups different types of variables under one name.

Useful to represent real-world entities like student, employee, car, etc.

1. Structure Declaration

```
struct Student {
  int roll;
 char name[50];
 float marks;
};
```

2. Structure Initialization

```
struct Student s1 = {101, "Bhattji", 92.5};
```

3. Accessing Structure Members

Use dot operator (.) for access:

```
printf("Roll: %d", s1.roll);
printf("Name: %s", s1.name);
printf("Marks: %.2f", s1.marks);
```

13- File handling allows a C program to store, read, write, and update data in files (on disk) instead of just using temporary memory (RAM).

It is important for:

Saving user data permanently

Reading configuration or logs

Working with large data files

Basic File Operations in C

Operation	Function Used
Open a file	fopen()
Read a file	fscanf(), fgets(), fgetc()
Write a file	fprintf(), fputs(), fputc()
Close a file	fclose()

1. Opening a File

```
FILE *fp;
fp = fopen("file.txt", "w");
```

2. Writing to a File

```
fprintf(fp, "Hello Bhattji!\n");
```

3. Reading from a File

```
char str[100];
fgets(str, 100, fp);   // Reads line

char ch = fgetc(fp);
```

4. Closing a File

```
fclose(fp);
```