Module 2 – Introduction to programming

Overview of C Programming

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

Answer ->

Overview of C Programming

The History and Evolution of C Programming

C programming is one of the most important and influential programming languages in the history of computer science. It was developed in the early 1970s by Dennis Ritchie at Bell Labs. C was created as an improvement over the B programming language and was designed to write the Unix operating system. This made C one of the first languages used for system-level programming.

Over time, C became popular because of its simplicity, speed, and power. It allows direct control over hardware and memory, which makes it perfect for writing operating systems, embedded systems, and performance-critical applications.

In the 1980s, C was standardized by ANSI (American National Standards Institute), creating what is known as ANSI C. This helped developers write portable code that could run on different machines with little or no changes. Later, more modern versions such as C99 and C11 added new features to make C even more flexible.

C has also influenced many other languages. Popular languages like C++, Java, C#, and even Python have roots in C's structure and syntax. This shows how strong and long-lasting its design is.

Why C is Still Important Today

Even today, C is widely used for several reasons:

- Efficiency**:** C programs are fast and use system resources well-
- Control: It gives programmers low-level access to memory and hardware.
- Portability**:** C code can be easily adapted to different platforms**.**

- Foundation for Learning**:** Learning C helps build a strong foundation in programming logic and system understanding.

Many modern systems like operating systems (Linux, Windows parts), embedded devices, and game engines still use C. It's also used in teaching computer science fundamentals.

2) LAB EXERCISE:

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

Answer -> Here are three real-world applications where **C programming** is extensively used:

1. Embedded Systems:
   C is widely used to program microcontrollers in devices like washing machines, medical instruments, and automotive systems due to its low-level hardware access and efficiency.
2. Operating Systems:
   Major operating systems like Windows, Linux, and macOS are developed primarily in C because it offers direct memory management and high performance.
3. Game Development:
   C is used in game engines (like Unreal Engine) and for developing performance-critical game components, especially for consoles and real-time rendering.

Setting Up Environment

3) THEORY EXERCISE:

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.

Answer ->

1. Install GCC Compiler (Windows using MinGW):

- Download **MinGW** from https://osdn.net/projects/mingw/.

- Run installer and select **mingw32-gcc-g++**.
- Add **C:\MinGW\bin** to **System PATH**.

2. Set Up an IDE:

a) DevC++:

- Download from https://sourceforge.net/projects/orwelldevcpp/.
- Install and open DevC++.
- GCC comes pre-installed.

b) Code::Blocks:

- Download from https://www.codeblocks.org/downloads/.
- Choose version with MinGW included.
- Install and start coding.

c) VS Code:

- Download from https://code.visualstudio.com/.
- Install the C/C++ extension from Microsoft.
- Install GCC separately (via MinGW).
- Configure tasks.json and launch.json for build/run.

4) LAB EXERCISE:

Install a C compiler on your system and configure the IDE. Write your first program to print "Hello, World!" and run it.

I installed a C compiler (GCC) on my system and configured the IDE (DevC++/CodeBlocks/VS Code).
Then, I wrote and executed my first C program:

```
#include <stdio.h>

int main() {

   printf("Hello, World!");
```

```
    return 0;
}
```

Basic Structure of a C Program

5) THEORY EXERCISE:

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 typical C program has the following components:

1. Header Files

2. Main Function

3. Comments

4. Data Types

5. Variables

1. Header Files

Header files tell the compiler to include libraries that provide built-in functions**.**

```
#include <stdio.h>  // for input and output functions like printf()
```

2. Main Function

Every C program starts from the main() function.

```
int main() {
    // Code goes here
    return 0;
}
```

- int means the function returns an integer.
- return 0; tells the operating system the program ended successfully.

3. Comments

Comments are notes in the code to explain what's happening. They are ignored by the compiler.

// This is a single-line comment

/* This is

a multi-line comment */

4. Data Types

Data types define what kind of data a variable can hold.

Data Type	Description	Example
Int	Integer values	5, -10
Float	Decimal values	3.14, -0.5
Char	Single characters	'a', 'Z'
String	Multiple characters	“hello world”

5. Variables

Variables are containers for storing data. **You must declare the type** before using them.

```
int age = 20;        // Integer variable
float weight = 65.5;  // Float variable
char grade = 'A';     // Character variable
```

Full Example Program

```
#include <stdio.h>  // Include standard input/output library
```

```
int main() {
   // Variable declarations
   int age = 20;
   float height = 5.9;
   char grade = 'A';

   // Output
   printf("Age: %d\n", age);
   printf("Height: %.1f\n", height);
   printf("Grade: %c\n", grade);

   return 0;  // End of program
}
```

Output:

```
Age: 20
Height: 5.9
Grade: A
```

6) LAB EXERCISE:

Write a C program that includes variables, constants, and comments. Declare and use different data types (int, char, float) and display their values.

Answer->

```
#include <stdio.h>
int main() {
   // Constant declaration
     const int YEAR = 2025;
```

```
   // Variable declarations
     int marks = 85;
     char section = 'B';
     float percentage = 84.75;
   // Output values
       printf("Student Details:\n");
      printf("Year: %d\n", YEAR);
      printf("Marks: %d\n", marks);
      printf("Section: %c\n", section);
      printf("Percentage: %.f%\n", percentage);
   return 0;
}
```

Operators in C

7) THEORY EXERCISE:

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

1. Arithmetic Operators

Used to perform basic mathematical operations.

Operator	Description	Example
+	Addition	a + b
-	Subtraction	a - b
*	Multiplication	a * b
/	Division	a / b
%	Modulus	a % b

2. Relational Operators

Used to compare two values or expressions.

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

3. Logical Operators

Used to combine multiple conditions.

Operator	Description	Example
&&	Logical AND	(a > 0 && b > 0)
||	Logical Or	(a > 0 && b > 0)
!	Logical NOT	!(a > b)

4. Assignment Operators

Used to assign values to variables.

Operator	Description	Example
=	Assign	a = 10
+=	Add and assign	a += 5
-=	Subtract and assign	a -= 2

Operator	Description	Example
*=	Multiply and assign	a *= 3
/=	Divide and assign	a /= 4

5. Increment/Decrement Operators

Used to increase or decrease a variable's value by 1.

Operator	Description	Example
++	Increment	a++ or ++a
--	Decrement	a-- or --a

6. Bitwise Operators

Used to perform bit-level operations.

Operator	Description	Example
&	AND	a & b
`	`	OR
^	XOR	a ^ b
~	NOT (1's complement)	~a
<<	Left shift	a << 2
>>	Right shift	a >> 2

7. Miscellaneous Operators

This concludes the notes on operators in C for your assignment.

1 sizeof
Returns size (in bytes) of a data type or variable.
Example: sizeof(int)

2. Ternary Operator ?:
Short if-else for quick decisions.
Example: max = (a > b) ? a : b;

3. Comma Operator ,
Evaluates multiple expressions, returns last.
Example: x = (1, 2, 3); // x = 3

4. Pointer Operators * and &
* dereferences pointer, & gets address.
Example: int *p = &a;

printf("%d", *p);

5 Member Access Operators . and ->
Access struct members via object or pointer.
Example: s.id and ptr->id

8) LAB EXERCISE:

Write a C program that accepts two integers from the user and performs arithmetic, relational, and logical operations on them. Display the results

```
#include <stdio.h>

int main() {

   int a, b;


   // Input two integers

   printf("Enter two integers: ");

   scanf("%d %d", &a, &b);


   // Arithmetic Operations

   printf("\n--- Arithmetic Operations ---\n");

   printf("a + b = %d\n", a + b);

   printf("a - b = %d\n", a - b);
```

```
    printf("a * b = %d\n", a * b);
    if (b != 0) {
        printf("a / b = %d\n", a / b);
        printf("a %% b = %d\n", a % b);
    } else {
        printf("Division and modulus by zero is not allowed.\n");
    }
    // Relational Operations
         printf("\n--- Relational Operations ---\n");
         printf("a == b: %d\n", a == b);
         printf("a != b: %d\n", a != b);
         printf("a > b : %d\n", a > b);
         printf("a < b : %d\n", a < b);
        printf("a >= b: %d\n", a >= b);
        printf("a <= b: %d\n", a <= b);
    // Logical Operations
    printf("\n--- Logical Operations ---\n");
    printf("a && b: %d\n", a && b);
    printf("a || b: %d\n", a || b);
    printf("!a    : %d\n", !a);
    printf("!b    : %d\n", !b);
    return 0;
}
--- Arithmetic Operations ---
a + b = 170
```

a - b = 10

a * b = 7200

a / b = 1

a % b = 10

--- Relational Operations ---

a == b: 0

a != b: 1

a > b : 1

a < b : 0

a >= b: 1

a <= b: 0

--- Logical Operations ---

a && b: 1

a || b: 1

!a : 0

!b : 0

Control Flow Statements in C

9) THEORY EXERCISE:

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

1. if statemnet

Executes a block only if the condition is true

Syntax:

```
if (condition) {
  // code}
```

Example:

```
int temp = 40;
if (temp > 35) {
       printf("It’s a hot day!\n");
}
```

2. if-else Statement

Chooses between two blocks based on a condition

Syntax:

```
if (condition) {
      // true block
            } else {
       // false block
 }
```

Example

```
int marks = 65;
if (marks >= 50) {
   printf("You passed the test.\n");
} else {
   printf("You failed the test.\n");
}
```

3. Nested if-else

An if or else inside another if-else, used for multiple conditions.

Syntex :

```
If (cond1) {
  if (cond2) {
```

```
      // block
  } else {
      // block
  }
} else {
  // block
}
```

Example

```
int age = 20;
int hasID = 1;
if (age >= 18) {
   if (hasID) {
      printf("You can vote.\n");
   } else {
      printf("ID required to vote.\n");
   }
} else {
   printf("You are underage.\n");
}
```

4. switch Statement

Selects from multiple options based on a value.

Syntex:

```
switch (variable) {
  case value1:
                code;
```

```
                break;
  case value2:
                code;
                break;
  default:
              // code;
}
```

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("Midweek Boost!\n");
         break;
      default:
         printf("Weekend vibe!\n");
   }
   return 0;
```

}

10) LAB EXERCISE:

Write a C program to check if a number is even or odd using an if-else statement. Extend the program using a switch statement to display the month name based on the user's input (1 for January, 2 for February, etc.)

```
#include <stdio.h>
int main() {
   int number, month;
   printf("Enter a number to check even or odd: ");
   scanf("%d", &number);
   if (number % 2 == 0) {
      printf("%d is Even.\n", number);
   } else {
      printf("%d is Odd.\n", number);
   }
   printf("Enter a number (1-12) to display the month name: ");
   scanf("%d", &month);
   switch(month) {
      case 1:
         printf("January\n");
         break;
      case 2:
         printf("February\n");
         break;
      case 3:
         printf("March\n");
```

```
    break;
case 4:
    printf("April\n");
    break;
case 5:
    printf("May\n");
    break;
case 6:
    printf("June\n");
    break;
case 7:
    printf("July\n");
    break;
case 8:
    printf("August\n");
    break;
case 9:
    printf("September\n");
    break;
case 10:
    printf("October\n");
    break;
case 11:
    printf("November\n");
    break;
```

```
        case 12:
            printf("December\n");
            break;
        default:
            printf("Invalid month number! \n");
            break;
    }
    return 0;
}
```

Looping in C

10) THEORY EXERCISE:

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

Following Are The Loops

1. While Loop

Syntax:

```
while(condition) {
  // code
}
```

- Condition is checked before the loop runs.
- Used when the number of iterations is not known in advance.

Example Use**:** Reading input until user types "exit".

2 For Loop

Syntax:

```
for(initialization; condition; increment/decrement) {
```

```
  // code
}
```

- Best when the number of iterations is known.
- Initialization, condition, and update are in one line.

Example Use**:** Looping from 1 to 10.

3 Do-While Loop

Syntax:

```
do {
  // code
} while(condition);
```

- Runs the code at least once, even if condition is false.
- Condition is checked after the loop body.

Example Use**:** Menu-driven programs where the menu shows at least once.

Summary Table:

Loop Type	Condition Check	Runs at least once?	Best Use Case
While	Before loop	No	Unknown iterations
For	Before loop	No	Known number of iterations
do-while	After loop	Yes	Run once, then check condition

11) LAB EXERCISE:

Write a C program to print numbers from 1 to 10 using all three types of loops (while, for, do-while).

1.While

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

```
int main() {
   int i;
       // Using while loop
                printf("Using while loop:\n");
                i = 1;
              while(i <= 10) {
                              printf("%d ", i);
                               i++;
               }
}
```

2 For Loop

```
#include <stdio.h>
   int main() {
                printf("\n\nUsing for loop:\n");
                 for(int i = 1; i <= 10; i++) {
                                 printf("%d ", i);
                   }
                }
```

3 do.. while

```
#include<stdio.h>
         Int main(){
                        printf("\n\nUsing do-while loop:\n");
                        int i = 1;
                        do {
```

```
                printf("%d ", i);
                i++;
                } while(i <= 10);
        }
```

13) THEORY EXERCISE

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

1 break Statement

Use -> It is used to exit from a loop or switch case immediately

Example ->

```
#include <stdio.h>
int main() {
   for (int i = 1; i <= 5; i++) {
        if (i == 3)
            break;  // loop stops when i = 3
        printf("%d ", i);
   }
   return 0;
}
```

2. continue Statement

Use: It is used to skip the current iteration and go to the next iteration of the loop.

Exmple ->

```
#include <stdio.h>
int main() {
   for (int i = 1; i <= 5; i++) {
```

```
        if (i == 3)
            continue;   // skip when i = 3
        printf("%d ", i);
    }
    return 0;
}
```

3 goto Statement

Use : It is used to jump to a specific labeled part of the code.

Example ->

```
#include <stdio.h>
int main() {
    int i = 1;
    if (i == 1)
        goto hello;  // jump to label

    printf("This won't print.\n");
hello:
    printf("Hello from goto!\n");
    return 0;
}
```

14) LAB EXERCISE:

Write a C program that uses the break statement to stop printing numbers when it reaches 5. Modify the program to skip printing the number 3 using the continue statement.

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

```
int main() {
   int i;
   for (i = 1; i <= 10; i++) {
       if (i == 3) {
           continue; // Skip printing 3
       }
       if (i == 5) {
           break;    // Stop loop when i is 5
       }
        printf("%d ", i);
   }
   return 0;
}
```

15) THEORY EXERCISE:

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.

It helps in reusing code and making the program more organized and easy to understand .

Types of Function Parts:

1. Function Declaration

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

```
return_type function_name(parameter_list);
```

2. Function Definition

Contains the actual body/code of the function

```
return_type function_name(parameter_list) {
```

```
        // code to execute
    }
```

3 Function Call

Used to execute the function

```
    function_name(arguments);
```

Example ->

```
#include <stdio.h>
    // Function Declaration
 void greet();
  int main() {
      printf("Before function call.\n");
       // Function Call
     greet();
     printf("After function call.\n");
     return 0;
  }
      // Function Definition
    void greet() {
       printf("Hello! Welcome to C programming.\n");
  }
```

17) LAB EXERCISE:

Write a C program that calculates the factorial of a number using a function. Include function declaration, definition, and call.

```
#include <stdio.h>
   // Function Declaration
```

```
    int factorial(int n);
int main() {
      int number, result;
            printf("Enter a number: ");
            scanf("%d", &number);
   // Function Call
   result = factorial(number);
   printf("Factorial of %d is %d\n", number, result);
   return 0;
}
// Function Definition
int factorial(int n) {
   int fact = 1;
   for (int i = 1; i <= n; i++) {
      fact = fact * i;
   }
   return fact;
}
```

18) THEORY EXERCISE:

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

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

- It allows you to store multiple values using a single variable name.
- Each element is accessed using an index, starting from 0.

1 One-Dimensional Array

A 1D array is like a list of values.

Example Of Use case ->stores marks of 5 students

Syntax

```
    data_type array_name[size];
```

Exmple

```
#include <stdio.h>
int main() {
   int numbers[5] = {10, 20, 30, 40, 50};
   for (int i = 0; i < 5; i++) {
      printf("%d ", numbers[i]);
   }
   return 0;
}
```

2 Multi-Dimensional Array.

A multi-dimensional array is an array of arrays, like a table or matrix.

Example Use Case

Syntax

```
data_type array_name[rows][columns];
```

```
#include <stdio.h>
int main() {
   int matrix[2][3] = {
      {1, 2, 3},
      {4, 5, 6}
   };
```

```
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 3; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

18 LAB EXERCISE:

Write a C program that stores 5 integers in a one-dimensional array and prints them. Extend this to handle a two-dimensional array (3x3 matrix) and calculate the sum of all elements.

```
#include <stdio.h>
int main() {
    // 1. One-Dimensional Array
    int arr[5] = {10, 20, 30, 40, 50};
    printf("One-Dimensional Array Elements:\n");
    for (int i = 0; i < 5; i++) {
        printf("%d ", arr[i]);
    }
    // 2. Two-Dimensional Array (3x3 Matrix)
    int matrix[3][3] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };
```

```
    int sum = 0;
    printf("\n\nTwo-Dimensional Array (3x3 Matrix):\n");
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            printf("%d ", matrix[i][j]);
            sum += matrix[i][j];  // Add each element to sum
        }
        printf("\n");
    }
    printf("\nSum of all elements in 2D array = %d\n", sum);
    return 0;
}
```

19) THEORY EXERCISE:

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

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

Instead of holding a value directly, it holds the memory location of a variable.

Pointers are very powerful in C and are used in arrays, functions, and dynamic memory.

Declration

```
    data_type *pointer_name;
```

* is used to declare a pointer.

data_type is the type of variable the pointer will point to

Initialization

```
int x = 10;
int *p = &x;
```

- &x gives the address of x, which is stored in pointer p

Example

```
#include <stdio.h>
int main() {
   int x = 10;
   int *ptr;
   ptr = &x;  // Pointer initialized with address of x
     printf("Value of x = %d\n", x);
     printf("Address of x = %p\n", &x);
     printf("Value stored in ptr (address) = %p\n", ptr);
     printf("Value pointed by ptr = %d\n", *ptr);  // dereferencing
   return 0;
}
```

20) LAB EXERCISE:

Write a C program to demonstrate pointer usage. Use a pointer to modify the value of a variable and print the result.

```
#include <stdio.h>

int main() {
   int num = 10;
   int *ptr;
   ptr = &num; // pointer stores address of num
   printf("Original value of num: %d\n", num);
   // Modify value using pointer
   *ptr = 25;
```

```
    printf("Modified value of num using pointer: %d\n", num);
    return 0;
}
```

21) THEORY EXERCISE:

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

These functions are available in the string.h header file and are used to perform various operations on strings.

1 strlen() – String Length

Use: Returns the length of a string (excluding the null \0 character).

Example

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

int main() {
    char str[] = "Hello";
    printf("Length = %lu\n", strlen(str));
    return 0;
}
```

2 . strcpy() – String Copy

- Use: Copies one string into another.

Example:

```
#include <stdio.h>
#include <string.h>
int main() {
    char src[] = "World";
```

```
    char dest[20];
    strcpy(dest, src);
    printf("Copied String = %s\n", dest);
    return 0;
}
```

3. strcat() – String Concatenation

- Use: Adds (appends) one string to the end of another.

Example:

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

int main() {
     char str1[20] = "Hello ";
     char str2[] = "World";
    strcat(str1, str2);
    printf("Concatenated String = %s\n", str1);
    return 0;
}
```

4 strcmp() – String Compare

Use: Compares two strings.

Returns 0 if equal.

Returns <0 if first < second.

Returns >0 if first > second.

Example

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

```
#include <string.h>
int main() {
    char a[] = "apple";
    char b[] = "apple";
    if (strcmp(a, b) == 0) {
        printf("Strings are equal\n");
    } else {
        printf("Strings are not equal\n");
    }
    return 0;
}
```

5 strchr() – Find Character in String

Use: Finds the first occurrence of a character in a string.

```
#include <stdio.h>
#include <string.h>
int main() {
    char str[] = "Welcome";
    char ch = 'c';
    char *ptr = strchr(str, ch);
    if (ptr != NULL) {
        printf("Character '%c' found at position: %ld\n", ch, ptr - str);
    } else {
        printf("Character '%c' not found in the string.\n", ch);
    }
    return 0;
```

```
}
```

22) LAB EXERCISE:

Write a C program that takes two strings from the user and concatenates them using strcat(). Display the concatenated string and its length using strlen().

```
#include <stdio.h>

#include <string.h>

int main() {

   char str1[100], str2[100];

     // Input two strings

   printf("Enter first string: ");

   scanf("%s", str1);

   printf("Enter second string: ");

   scanf("%s", str2);

     // Concatenate str2 to str1

   strcat(str1, str2);

    // Display result

   printf("Concatenated string: %s\n", str1);

   printf("Length of concatenated string: %d\n", strlen(str1));

   return 0;

}
```

23) THEORY EXERCISE:

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

A structure in C is a user-defined data type that allows you to group different types of variables under one name.

- It is used to represent a record.

- Useful when you want to store related data (like name, age, and marks of a student).

How To Declare Structure

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

How to initialize a structure

```
struct Student s1 = {"Rahul", 20, 85.5};
```

You can also assign members later:

```
struct Student s2;
strcpy(s2.name, "Priya"); // use string.h for strcpy
s2.age = 22;
s2.marks = 90.0;
```

How to Access Structure Members

Use the dot (.) operator:

```
printf("Name: %s\n", s1.name);
printf("Age: %d\n", s1.age);
printf("Marks: %.2f\n", s1.marks);
```

Example

```
#include <stdio.h>
#include <string.h>
struct Student {
   char name[50];
```

```
    int age;
    float marks;
};
int main() {
    struct Student s;
    strcpy(s.name, "Amit");
    s.age = 21;
    s.marks = 88.5;
    printf("Student Information:\n");
    printf("Name: %s\n", s.name);
    printf("Age: %d\n", s.age);
    printf("Marks: %.2f\n", s.marks);
    return 0;
}
```

24) LAB EXERCISE:

Write a C program that defines a structure to store a student's details (name, roll number, and marks). Use an array of structures to store details of 3 students and print them.

```
#include <stdio.h>
#include <string.h>
struct Student {
    char name[50];
    int roll;
    float marks;
};
int main() {
```

```
    struct Student s[3];
    for (int i = 0; i < 3; i++) {
        printf("Enter details for Student %d:\n", i + 1);
        printf("Name: ");
        scanf("%s", s[i].name);  // Reads name (no spaces)
        printf("Roll Number: ");
        scanf("%d", &s[i].roll);
        printf("Marks: ");
        scanf("%f", &s[i].marks);
    }
    printf("\n--- Student Details ---\n");
    for (int i = 0; i < 3; i++) {
        printf("Student %d:\n", i + 1);
        printf("Name: %s\n", s[i].name);
        printf("Roll Num : %d\n", s[i].roll);
        printf("Marks: %.2f\n", s[i].marks);
        printf("\n");
    }
    return 0;
}
```

25) THEORY EXERCISE: Explain the importance of file handling in C. Discuss how to perform file operations like opening, closing, reading, and writing files.

In C, file handling allows a program to store data permanently on the disk instead of just in memory (RAM).

Without file handling, all data is lost once the program ends.

Why File Handling is Important:

- Permanent storage of data
- Input/output from files (instead of keyboard/screen)
- Handling large amounts of data
- Reading/writing structured data (e.g., records)

1 Opening a File

Use the fopen function

```
FILE *fp;
fp = fopen("file.txt", "w");  // Opens file for writing
```

Mode	Meaning
"r"	Read (file must exist)
"w"	Write (creates new file or erases old)
"a"	Append (add to end)
"r+"	Read and write
"w+"	Write and read
"a+"	Append and read

2 Writing to a File

Use fprintf() or fputs():

```
fprintf(fp, "Hello World!");
```

3 Reading from a File

Use fscanf() or fgets():

```
char text[100];
fscanf(fp, "%s", text);  // reads a word
```

4 Closing a File

Use fclose() to close the file:

```
fclose(fp);
```

Example Program: Write and Read a File

```
#include <stdio.h>
int main() {
   FILE *fp;
   fp = fopen("data.txt", "w");
   fprintf(fp, "This is a test file.");
   fclose(fp);
   char str[100];
   fp = fopen("data.txt", "r");
   fscanf(fp, "%[^\n]", str);  // read until newline
   printf("File Content: %s\n", str);
   fclose(fp);
   return 0;
}
```

26) LAB EXERCISE:

Write a C program to create a file, write a string into it, close the file, then open the file again to read and display its contents.

```
#include <stdio.h>

int main() {
   FILE *fp;
   char str[100] = "This is a file handling example in C.";
   char buffer[100];
```

```
    // Step 1: Create and write to the file
    fp = fopen("sample.txt", "w");  // open file for writing
    if (fp == NULL) {
        printf("Error creating file.\n");
        return 1;
    }
    fprintf(fp, "%s", str);  // write string to file
    fclose(fp);  // close file

    // Step 2: Open and read from the file
    fp = fopen("sample.txt", "r");  // open file for reading
    if (fp == NULL) {
        printf("Error opening file.\n");
        return 1;
    }
    fgets(buffer, 100, fp);  // read from file
    printf("File content: %s\n", buffer);  // display content
    fclose(fp);  // close file

    return 0;
}
```