Module 2 – Introduction to Programming

1.Overview of C Programming

THEORY EXERCISE:

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

------C programming was developed by Dennis Ritchie at Bell Labs in 1972 as an evolution of the B language, primarily for system programming. It gained popularity through its use in developing the Unix operating system. In the 1980s, it was standardized as ANSI C, and over time, updates like C99 and C11 introduced new features. C remains a foundational language, influencing many modern languages and widely used in system-level and performance-critical applications.

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.
------Applications of C:-
1. Operating System Development
2. Embedded Systems and IoT Devices
3. Compilers and Interpreters
4. Database Systems
2.Setting Up Environment :-

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

```
#include<stdio.h>
void main(){
printf(“Hello World”);
}
```

3. **Basic Structure of a C Program**

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

Basic Stucture Of C:-
Structure of a C Program a C program typically consists of three main parts:

1. **Preprocessor Directives: Instructions like #include to include libraries, which are processed before compilation.**
2. **2. Main Function: The entry point of the program, written as int main(), where execution begins.**
3. **Functions and Statements: The body of the program containing logic and functions to perform specific tasks, with the main function returning an integer value (return 0;).**

Comments:-
Comments in C In C,
comments are used to add explanatory notes within the code, which are ignored by the compiler. There are two types of comments:

1. **Single-line comments: Use // to comment out a single line.**
2. **Multi-line comments: Enclosed between /* and */, used for comments spanning multiple lines**

Data Types
• A data type specifies what type of data a variablecan store such as integer, floating, character, etc. int,float, char
Example int res = 5;

Variables
• Variables are the names you give to computer memory locations which are used to store values in a computer program.
• For example, assume you want to store two values 10 and 20 in your program and at a later stage, you want to use these two values. Let's see how you will do it. Here are the following three simple steps

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.

#include <stdio.h>

```
int main()
{
   // Declare a constant
   const float PI = 3.14;

   // Declare variables of different data types
   int age = 20;                  // Integer variable
   char grade = 'A';              // Character variable
   float height = 5.9;            // Floating-point variable

   // Print the values of variables and constant
   printf("Age: %d\n", age);
   printf("Grade: %c\n", grade);
   printf("Height: %.1f feet\n", height);
   printf("Value of PI (constant): %.2f\n", PI);

   return 0;
}
```

4. Operators in C

THEORY EXERCISE:

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

Arithmetic Operators:-

Arithmetic Operators are used to perform operations likeaddition,multiplication,division,subsraction,moduloetc..

Operator	Function	Example
+	Addition	a+b
-	Substaction	a-b
*	Multiplication	a*b
/	Division	a/b
%	Modulo	a%b

Example of Arithmetic Operator:-

```
#include<stdio.h>
int main(){
int a=5,b=10;
printf("Additon is:%d",a+b);
printf("Substraction is:%d",a-b);
printf("Multiplication is:%d",a*b);
printf("Division is:%d",a/b);
return 0;
}
```

Relational Operators:-
When we have to perform only one task or we have onle one condition when we use relation operators.

Operator	Meaning Of Operator	Example
<	Less then	a < b
>	Greater then	a > b
<=	Less then equall to	a<=b
>=	Greater then equall to	a>=b
= =	Equal to	a==b
!=	Not equall to	a!=b

Logical Operators:-
when we have multiple conditions then we have to use relation operators.

Operators	Meaning	Example
&&	Logical AND. True only if all operands are true	If c = 5 and d = 2 then, expression ((c==5) && (d>5)) equals to 0.
||	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression ((c==5) || (d>5)) equals to 1.
!	Logical NOT. True only if the operand is 0	If c = 5 then, expression !(c==5) equals to 0.

Assignment Operators:-

The Basic type of Assignment operator is '='.

Operators	Example
+=	a+=b -->a=a+b
-=	a-=b -->a=a-b
*=	a*=b -->a=a*b
/=	A/= -->a=a/b

Bitwise Operators:-

Operators	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

conditional operators:-
The conditional operator in C, also known as the ternary operator, is a concise way to express conditional statements. It operates on three operands and is represented by the symbols ? And :. This operator is often used as a shorthand for the if-else statement, allowing for more compact code.

variable = (condition) ? expression_if_true : expression_if_false;

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 num1, num2;

   // Accept two integers from the user
   printf("Enter the first integer: ");
   scanf("%d", &num1);
   printf("Enter the second integer: ");
```

```
    scanf("%d", &num2);

    // Arithmetic operations
    printf("\nArithmetic Operations:\n");
    printf("%d + %d = %d\n", num1, num2, num1 + num2);
    printf("%d - %d = %d\n", num1, num2, num1 - num2);
    printf("%d * %d = %d\n", num1, num2, num1 * num2);
    if (num2 != 0) {
        printf("%d / %d = %d\n", num1, num2, num1 / num2);
        printf("%d %% %d = %d\n", num1, num2, num1 % num2);
    } else {
        printf("Division and modulus by zero are undefined.\n");
    }

    // Relational operations
    printf("\nRelational Operations:\n");
    printf("%d == %d: %d\n", num1, num2, num1 == num2);
    printf("%d != %d: %d\n", num1, num2, num1 != num2);
    printf("%d > %d: %d\n", num1, num2, num1 > num2);
    printf("%d < %d: %d\n", num1, num2, num1 < num2);
    printf("%d >= %d: %d\n", num1, num2, num1 >= num2);
    printf("%d <= %d: %d\n", num1, num2, num1 <= num2);

    // Logical operations
    printf("\nLogical Operations:\n");
    printf("(%d && %d): %d\n", num1, num2, num1 && num2);
    printf("(%d || %d): %d\n", num1, num2, num1 || num2);
    printf("!%d: %d\n", num1, !num1);
    printf("!%d: %d\n", num2, !num2);

    return 0;
}result
```

5. **Control Flow Statements in C**
   THEORY EXERCISE:
   Explain decision-making statements in C (if, else, nested if-else, switch). Provide examples of each.
   ------ if:-
   if statement is the basic decision making statement
   Used to decide whether a certain statement or block of statements will be executed or not.

- **Syntax :**

```
if( condition )
{
    statement_1 ; // true block statements
}
statement x;
```

Example:-

```
#incude<stdio.h>
#incude<conio.h>

void main()
{
  int  a= 3,b=5;
  if(a < b){
    printf("a less then b);
    }
 }
```

If else Statements:-

- **if else statement allows selecting any one of the two available options depending upon the output of the test condition**

Syntax :

```
if (condition )
{
      statements;
}
 else
{
 statements ; // false statement
}
```

Example:-

```
#include<stdio.h>
int main()
{
int a=5,b=4;
if(a>b)
{
  printf(“a is greater than b”);
}
else
```

```
{
printf("a is less than b");
}
return 0;
}
```

Nested If Statements

Nested if statement is simply an if statement embedded with an another if statement

Syntax :

```
 if (condition1)
{
   statements ;
     if ( condition2)
   {
   statements ;
   }
}
```

Example:-

```
#include<stdio.h>
int main()
{
int age=19;
int l=1;
if(age>18)
{
    printf("congratuations ! You are eligible for drive\n");
    if(l==1)
        {
            printf("Greate ! You can drive safely");
         }
     else
        {
           printf("You are eligible but you do not prosess a licence");
        }
}
else
{
printf("You are not eligible for drive");
}
```

return 0;
}

Switch Statements:-
Switch case statements are a substitute for long if statements that compare a variable to several integer values
Syntax :

```
switch ( n)
{
    case 1 :
         break;
    case 2 :
        break ;
    default :
}
```

Example:-

```
#include<stdio.h>
int main()
{
 int i=2;
 switch(i)
{
  case 1:
  printf("case 1");
  break;
  case 2:
  printf("case 2");
  break;
  case 3:
   printf("case 3");
   break;
   default:
    printf("default");
    break;
}
return 0;
}
```

6. **Looping in C**
   - **THEORY EXERCISE:**

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

----A loop statement allows us to execute a statement or group of statements multiple times based on a condition.
--- for loops:-
When you know exactly how many times you want to loop through a block of code, use the for loop instead of a while loop:
---Syntax:-

```
for(expression 1; expression 2; expression 3) {
// code block to be executed
}
```

in for we have three parameters:
1.Initilization:Where we have to start
2.condition or Ending:-where we have to stop
3.Increment-Decrement:where we have to go means,use ++ operator for increment– operator for decrement.

Example:-

```
#include<stdio.h>
void main(){
for(int i=1;i<=10;i++)
{
printf(“Hello\n”);
}
printf(“hey”);
}
```

---While loops:-
The while loop loops through a block of code as long as a specified condition is true
Syntax:

```
while (condition) {
// code block to be executed
}
```

example:--

```
#include<stdio.h>
void main(){
```

Do while loops:-
The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.
Syntax:-

```
do{
// code block to be executed
}
while
(condition);
```

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

Using a for loop

```
#include <stdio.h>

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

Using a while loop

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

```
    printf("\n");
}
```

Using a do-while loop

```
#include <stdio.h>

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

    return 0;
}
```

7. **Loop Control Statements**
   - **THEORY EXERCISE:**
   - **o Explain the use of break, continue, and goto statements in C. Provide examples of each.**

GOTO statment:-

• By using this goto statements we can transfer the control from current location to anywhere in the program.

• To do all this we have to specify a label with goto and the control will transfer to the location where the label is specified.

Example:-

```
#include<stdio.h>
 void main ()
{
    int n, sum = 0, i = 0 ;

    printf ("Enter a number") ;
    scanf ("%d", &n);

    loop:
    i++ ;
    sum += i ;
```

```
    if (i < n)
        goto loop;
    printf ("\n sum of %d natural numbers = %d", n, sum) ;
}
```

The Break Statement:

- **The break statement is used inside loop or switch statement.**
- **When compiler finds the break statement inside a loop, compiler will abort the loop and continue to execute statements followed by loop**

Syntax: break ;

Example:

```
#include<stdio.h>
 void main(){
  int i =0;
  while(1)
  {
    i=i+1;
    printf("the value of i is%d\n",i);
    if(i>5)
  {
      break;
    }
  }
}
```

Continue statement:

- **The continue statement is also used inside loop.**
- **When compiler finds the continue statement inside a loop, compiler will skip all the following statements in the loop and resume the next loop iteration.**

Syntax: continue ;

example:-

```
#include <stdio.h>
int main()
{
for (int i = 1; i <= 5; i++)
{
if (i == 3)
{ continue; // Skip the rest of the loop when i is 3
}
printf("Number: %d\n", i);
}
```

```
return 0;
}
```

- **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.**

```
// Print numbers from 1 to 10, but stop when number reaches 5
#include <stdio.h>

int main() {
 for (int i = 1; i <= 10; i++) {
if (i == 5) {
 break; // Stop the loop when i is 5
}
printf("%d\n", i);
 }
 return 0;
}

// Print numbers from 1 to 10, skip 3, stop at 5
#include <stdio.h>

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

8. **Functions in C**
   THEORY EXERCISE:
   o What are functions in C? Explain function declaration, definition, and how to call a function. Provide examples.

Functions in C are blocks of code designed to perform specific tasks. They help in organizing code, improving readability, and enabling reusability. A function typically consists of three main components:declaration,definition, and function call.

1.Function Declaration

A function declaration (or prototype) informs the compiler about the function's name, return type, and parameters. It is usually placed before the main() function or in a header file.

Syntax:

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

2.Function Definition

The function definition contains the actual code or logic of the function. It specifies what the function does when called.

Syntax:

```
return_type function_name(parameter_list) {
   // Function body
   return value; // Optional, depending on return type
}
```

Example:

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

3. Function Call

A function call is used to execute the function. You pass the required arguments to the function, and it returns the result (if applicable).

Syntax:

Copy the codefunction_name(arguments);

Example:

Copy the codeint result = add(5, 3); // Calls the 'add' function with arguments 5 and 3

Complete Example

Here’s a full program demonstrating function declaration, definition, and calling:

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

```
// Function declaration
int add(int a, int b);

int main() {
   int num1 = 10, num2 = 20;

   // Function call
   int sum = add(num1, num2);

   printf("The sum of %d and %d is %d\n", num1, num2, sum);

   return 0;
}

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

LAB EXERCISE:

o 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 num;
   printf("Enter a positive integer: ");
   scanf("%d", &num);
   if (num < 0) {
      printf("Factorial is not defined for negative numbers.\n");
   } else {
      printf("Factorial of %d is %d\n", num, factorial(num)); // Function
call
   }
 return 0;
}

// Function definition
int factorial(int n) {
   if (n == 0 || n == 1) {
      return 1; // Base case
```

```
    }
    return n * factorial(n - 1); // Recursive case
}
```

9. Arrays in C

• THEORY EXERCISE:

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

Concept of Arrays in C

An array in C is a collection of elements of the same data type stored in contiguous memory locations. Arrays allow you to store multiple values under a single variable name, making it easier to manage and manipulate data. Each element in an array is accessed using an index, starting from 0.

Key Features of Arrays:

- **•Fixed Size:The size of an array is defined at the time of declaration and cannot be changed dynamically.**
- **•Homogeneous Data:All elements in an array must be of the same data type.**
- **•Random Access:Elements can be accessed directly using their index.**

Parameters	One-Dimensional Array	Two-Dimensional Array
Basics	A one-dimensional array stores a single list of various elements having a similar data type.	A two-dimensional array stores an array of various arrays,or a list of various lists, or an array of various one-dimensional arrays.
Representation	It represents multiple data items in the form of a list.	It represents multiple data items in the form of a table that contains columns and rows.
Dimensions	It has only one dimension.	It has a total of two dimensions.
Parameters of Receiving	One can easily receive it in a pointer, an unsized array, or a sized array.	The parameters that receive it must define an array's rightmost dimension.
Total Size (in terms of Bytes)	Total number of Bytes = The size of array x the size of array variable or datatype.	Total number of Bytes = The size of array visible or datatype x the size of second index x the size of the first index.

Concept of Arrays in C

An array in C is a collection of elements of the same data type stored in contiguous memory locations. Arrays allow you to store multiple values under a single variable name, making it easier to manage and manipulate data. Each element in an array is accessed using an index, starting from 0.

Key Features of Arrays:

- **Fixed Size: The size of an array is defined at the time of declaration and cannot be changed dynamically.**
- **Homogeneous Data: All elements in an array must be of the same data type.**
- **Random Access: Elements can be accessed directly using their index.**

Difference Between One-Dimensional and Multi-Dimensional Arrays

Aspect	One-Dimensional Array	Multi-Dimensional Array
Definition	A linear array with elements arranged in a single row.	An array with elements arranged in multiple rows and columns (or higher dimensions).
Structure	Single index is used to access elements.	Multiple indices are used to access elements.
Memory Representation	Stored in a single contiguous block.	Stored in a row-major order (rows stored sequentially).
Example	int arr[5];	int arr[3][4]; (2D) or int arr[2][3][4]; (3D).
Use Case	Suitable for simple lists like marks, prices, etc.	Suitable for matrices, tables, or grids.

Examples

1. One-Dimensional Array

```
#include <stdio.h>

int main() {
   int arr[5] = {10, 20, 30, 40, 50}; // Declaration and initialization

   // Accessing elements
   for (int i = 0; i < 5; i++) {
      printf("Element at index %d: %d\n", i, arr[i]);
   }

   return 0;
}
```

2. Multi-Dimensional Array (2D Array)

```
#include <stdio.h>

int main() {
   int matrix[2][3] = { {1, 2, 3}, {4, 5, 6} }; // 2 rows, 3 columns

   // Accessing elements
   for (int i = 0; i < 2; i++) {
      for (int j = 0; j < 3; j++) {
         printf("Element at [%d][%d]: %d\n", i, j, matrix[i][j]);
      }
   }

   return 0;
}
```

LAB EXERCISE:
o 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() {
   // One-dimensional array
   int arr[5] = {1, 2, 3, 4, 5};
   printf("One-dimensional array elements:\n");
   for (int i = 0; i < 5; i++) {
      printf("%d ", arr[i]);
   }
   printf("\n");

   // Two-dimensional array (3x3 matrix)
   int matrix[3][3] = {
      {1, 2, 3},
      {4, 5, 6},
      {7, 8, 9}
   };
   int sum = 0;
```

```
    printf("\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];
        }
        printf("\n");
    }

    printf("\nSum of all elements in the matrix: %d\n", sum);

    return 0;
}
```

10. Pointers in C

THEORY EXERCISE:

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

With pointers, you can access and modify the data located in the memory, pass the data efficiently between the functions, and create dynamic data structures like linked lists, trees, and graphs. To declare a pointer, use the dereferencing operator (*) followed by the data type.

LAB EXERCISE:

o 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 number = 10;  // Declare an integer variable
    int *ptr = &number;  // Declare a pointer and assign it the address of 'number'

    printf("Original value of number: %d\n", number);

    // Modify the value of 'number' using the pointer
    *ptr = 20;

    printf("Modified value of number using pointer: %d\n", number);
```

```
    return 0;
}
```

11.Strings in C

THEORY EXERCISE:

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

- **A string is an array of characters stored in a consecutive memory locations**
- **The ending character is always the null character '\0'. It acts as string terminator.**
- **Syntax : char string_name [length] ;**
- **The compiler automatically places the '\0' at the end of the string when we initializes the array**

1.strlen(String Length)

Use Case: To determine the length of a string for validation or allocation purposes.

Example:

Copy the code#include <stdio.h>

```
#include <string.h>

int main() {
    char name[] = "Ahmedabad";
    size_t length = strlen(name);
    printf("The length of the string is: %zu\n", length);
    return 0;
}
```

Scenario: Useful when you need to validate input length, e.g., ensuring a username is within a specific character limit.

2.strcpy(String Copy)

Use Case: To copy one string into another.

Example:

Copy the code#include <stdio.h>

```
#include <string.h>

int main() {
    char source[] = "Hello, World!";
    char destination[50];
```

```
    strcpy(destination, source);
    printf("Copied string: %s\n", destination);
    return 0;
}
```
Scenario: Useful when initializing or duplicating strings, such as copying user input into a buffer for further processing.

3.strcat(String Concatenation)

Use Case: To append one string to another.

Example:

Copy the code
```
#include <stdio.h>
#include <string.h>

int main() {
    char greeting[50] = "Hello, ";
    char name[] = "Alice!";
    strcat(greeting, name);
    printf("Concatenated string: %s\n", greeting);
    return 0;
}
```
Scenario: Useful for constructing messages, e.g., combining a greeting with a user's name.

4.strcmp(String Comparison)

Use Case: To compare two strings for equality or order.

Example:

Copy the code
```
#include <stdio.h>
#include <string.h>

int main() {
    char str1[] = "apple";
    char str2[] = "banana";
    if (strcmp(str1, str2) < 0) {
        printf("%s comes before %s alphabetically.\n", str1, str2);
    } else if (strcmp(str1, str2) > 0) {
        printf("%s comes after %s alphabetically.\n", str1, str2);
    } else {
        printf("The strings are identical.\n");
    }
    return 0;
}
```
Scenario: Useful for sorting strings or checking if two strings are identical, e.g., in a login system.

5.strchr(String Character Search)

Use Case: To find the first occurrence of a character in a string.
Example:
Copy the code#include <stdio.h>
#include <string.h>

```
int main() {
   char str[] = "Find the letter 'e' in this sentence.";
   char *result = strchr(str, 'e');
   if (result) {
      printf("First occurrence of 'e' is at position: %ld\n", result - str);
   } else {
      printf("'e' not found in the string.\n");
   }
   return 0;
}
```

12. Structures in C

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

A structure is a user-defined data type that can be used to group items of possibly different types into a single type. The struct keyword is used to define a structure. The items in the structure are called its member and they can be of any valid data type.

Declare a structure

```
struct student
{
   char  name[100];
   int   roll;
   float marks;
};
```

LAB EXERCISE: o 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
{
  int roll_no;
```

```
  char name[20];
  int marks1,marks2,marks3;
};
int main()
{
   struct student s1[5];
   int total[5] ,Choice ,j;
   char name[50];
   float Per[80];
   printf("Enter The Records Of 5 student's");
  for (int i=0;i<5;i++)
  {

   printf("Student of Detealis %d\n",i+1);
   printf("Enter student %d Roll_NO = ", i+1);
   scanf("%d",&s1[i].roll_no);
   printf("Enter student %d Name = ",i+1);
   scanf("%s",&s1[i].name);
   printf("Enter student %d MARKS_1 = ",i+1);
   scanf("%d",&s1[i].marks1);
   printf("Enter student %d MARKS_2 = ",i+1);
   scanf("%d",&s1[i].marks2);
   printf("Enter student %d MARKS_3 = ",i+1);
   scanf("%d",&s1[i].marks3);
  }

  printf("select 1 for Add More Record or select  to show Result or enter 2
for exit ");
  scanf("%d",&Choice);
  printf("\n");
  switch (Choice)
  {
  case 1:
       printf("ENTER NUMBER OF RECORD THAT YOU WANT TO
INSERT = ");
       scanf("%d",&j);
       for(int i=5;i<5+j;i++)
      {

       printf("Student of Detealis %d\n",i+1);
       printf("Enter student %d Roll_NO = ", i+1);
       scanf("%d",&s1[i].roll_no);
```

```
        printf("Enter student %d Name = ",i+1);
        scanf("%s",&s1[i].name);
        printf("Enter student %d MARKS_1 = ",i+1);
        scanf("%d",&s1[i].marks1);
        printf("Enter student %d MARKS_2 = ",i+1);
        scanf("%d",&s1[i].marks2);+
        printf("Enter student %d MARKS_3 = ",i+1);
        scanf("%d",&s1[i].marks3);
      }
      printf("Total Number Of Student Recored By The Adminstrator =
%d/n/n",5+j);
       for(int i=0;i<5+j;i++)
         {
            printf("  Detealis Student of %d\n",i+1);
            printf("STUDENTS ROLL NO = %d \n",s1[i].roll_no);
            printf("STUDENTS NAME = %s \n",s1[i].name);
            printf("STUDENTS MARKS_1 = %d \n",s1[i].marks1);
            printf("STUDENTS MARKS_2 = %d \n",s1[i].marks2);
            printf("STUDENTS MARKS_2 = %d \n\n\n,",s1[i].marks3);
            Per[i] = s1[i].marks1 + s1[i].marks2 + s1[i].marks3;
            printf("STUDENT'S PERSANTAGE= %.2f",Per[i]);

         }

      break;
case 1:
printf("Total Number Of Student Recored By The Adminstrator =
%d/n/n",5+j);
for(int i=0;i<5+j;i++)
{
printf(" Detealis Student of %d\n",i+1);
printf("STUDENTS ROLL NO = %d \n",s1[i].roll_no);
printf("STUDENTS NAME = %s \n",s1[i].name);
printf("STUDENTS MARKS_1 = %d \n",s1[i].marks1);
printf("STUDENTS MARKS_2 = %d \n",s1[i].marks2);
printf("STUDENTS MARKS_2 = %d \n\n\n,",s1[i].marks3);
Per[i] = (s1[i].marks1 + s1[i].marks2 + s1[i].marks3)/3;
printf("STUDENT'S PERSANTAGE= %.2f",Per[i]);
         }
         break;

   default:
```

```
        printf("Thank Yo So Much Thats The End Of Program");
    break;
  }


  return 0;
}
```

13.File Handling in C

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

File handling in C is a crucial aspect of programming that allows developers to store, retrieve, and manipulate data in files. This is especially important for applications that require persistent data storage, such as databases, configuration files, or logs. Here's an overview of its importance and how to perform basic file operations:

Importance of File Handling in C

1. **Data Persistence: Files enable data to persist beyond the program's execution, unlike variables stored in memory.**
2. **Large Data Management: Files allow handling large datasets that cannot fit into memory.**
3. **Data Sharing: Files facilitate data exchange between different programs or systems.**
4. **Logging and Debugging: Files are used to log program activities or errors for debugging purposes.**
5. **Backup and Recovery: Files provide a way to back up critical data for recovery in case of system failures.**

File Operations in C

C provides a set of standard library functions for file handling, which are included in the <stdio.h> header file. Below are the key operations:

1. Opening a File

- **Use the fopen() function to open a file.**
- **Syntax: FILE *fopen(const char *filename, const char *mode);**
- **Modes:**
  - **"r": Open for reading.**
  - **"w": Open for writing (creates a new file or truncates an existing file).**
  - **"a": Open for appending.**
  - **"r+", "w+", "a+": Open for both reading and writing.**

Example:

Copy the codeFILE *file = fopen("example.txt", "w");
if (file == NULL) {
printf("Error opening file.\n");
}
2. Closing a File
•Use the fclose() function to close a file.
•Syntax: int fclose(FILE *stream);
•Always close files to free resources and ensure data integrity.
Example:
Copy the codefclose(file);
3. Reading from a File
•Use functions like fgetc(), fgets(), or fread() to read data.
•fgetc(): Reads a single character.
•fgets(): Reads a string (line) from the file.
•fread(): Reads binary data.
Example:
Copy the codeFILE *file = fopen("example.txt", "r");
char buffer[100];
if (file != NULL) {
while (fgets(buffer, sizeof(buffer), file) != NULL) {
printf("%s", buffer);
}
fclose(file);
}
4. Writing to a File
•Use functions like fputc(), fputs(), or fwrite() to write data.
•fputc(): Writes a single character.
•fputs(): Writes a string.
•fwrite(): Writes binary data.
Example:
Copy the codeFILE *file = fopen("example.txt", "w");
if (file != NULL) {
fputs("Hello, World!\n", file);
fclose(file);
}
• 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>
#include <stdlib.h>

```
int main() {
    FILE *file;
    char str[] = "Hello, this is a sample string!";
    char buffer[100];

    // Create and write to the file
    file = fopen("example.txt", "w");
    if (file == NULL) {
        perror("Error opening file for writing");
        return 1;
    }
    fprintf(file, "%s", str);
    fclose(file);

    // Open and read from the file
    file = fopen("example.txt", "r");
    if (file == NULL) {
        perror("Error opening file for reading");
        return 1;
    }
    if (fgets(buffer, sizeof(buffer), file) != NULL) {
        printf("File contents: %s\n", buffer);
    }
    fclose(file);

return 0;
}
```