VSEC102

Anjuman — I — Islam's

M.H SABOO SIDDIK COLLEGE OF ENGINEERING

8, Saboo Siddik Polytechnic Rd., Byculla, Mumbai — 400 008.

Department of Basic Science and Humanity

Academic Year: 2025-26

C Programming Lab

NAME:

CLASS & Div:

ROLL NO.:

Anjuman — I — Islam's

M.H SABOO SIDDIK COLLEGE OF ENGINEERING

8, Saboo Siddik Polytechnic Rd., Byculla, Mumbai — 400 008.

Department of Basic Science and Humanity

Academic Year: 2025-26

Lab File

Class & Semester: *FE (Sem 1)*

Division: ……..

Subject: *C Programming Lab*

Name: ……

Roll No.: …….

CLASS & Div: …….

Document Revised on :*15 / 09 / 2025*

Anjuman — I — Islam's

M.H SABOO SIDDIK COLLEGE OF ENGINEERING

8, Saboo Siddik Polytechnic Rd., Byculla, Mumbai — 400 008.

Department of Basic Science and Humanity
Academic Year: 2025-26

Certificate

This is to certify that .....of the First-year Basic Science and Humanity Department, Division …… has performed all the experiments of C Programming Lab with satisfactory results during the academic year 2025-26.

…….. ……..

Name & Sign *Name & Sign*

Subject In-charge **H.o.D. ,** Department of Basic Science and Humanity

…….. ……..

Name & Sign *Name & Sign*

Internal Examiner **External Examiner**

Our Vision

To be an institute of global repute committed to the cause of nation building through technology-based education.

Our Mission

To be enabler of creation and dissemination of futuristic knowledge in technology through research and quality education.

PROGRAM OUTCOMES

PO1 : Engineering knowledge

PO2 : Problem analysis

PO3 : Design/development of solutions

PO4 : Conduct investigations of complex problems

PO5 : Modern tool usage

PO6 : The engineer and society

PO7 : Environment and sustainability

PO8 : Ethics

PO9 : Individual and team work

PO10 : Communication

PO11 : Project management and finance

PO12 : Life-long learning

Lab Objectives

1. Understand and use basic terminology in computer programming.
2. Use various data types in C programs effectively.
3. Design and implement programs involving decision structures, loops, and functions.
4. Design Implement Arrays , String, and Structure
5. Describe and utilize memory dynamics through the use of pointers.
6. Use different data structures and create/update basic data files in C.

Course Outcomes

Learner will be able to…

CO1. Illustrate the basic terminology used in computer programming.
CO2. Use different data types in a computer program.
CO3. Design programs involving decision structures, loops and functions.
CO4. Implement Arrays , String, and Structure
CO5. Describe the dynamics of memory by the use of pointers.
CO6. Use different data structures and create/update basic data files.

Rubrics

Following rubrics will be used to assess the work submitted by the students.

1. **For Experiments**

On Time Submission & Completion				
Defines tasks completion and submission				
	3	2	1	0
	Excellent	Good	Average	Poor
	All tasks are submitted and completed before or on the deadline consistently, showing strong time management skills.	Most tasks are submitted on or just before the deadline, with occasional minor delays. Time management is generally effective.	Several tasks are submitted late, but completion is mostly within an acceptable timeframe. Time management needs improvement.	Tasks are frequently submitted late or incomplete, showing poor time management and lack of adherence to deadlines.
Knowledge				
Defines Understanding of a student				
5	4	3	2	1
Excellent	Good	Satisfactory	Needs Improvement	Poor
Demonstrates thorough and deep understanding of the subject. Can accurately explain concepts and apply knowledge effectively.	Shows good understanding with minor gaps. Can explain most concepts clearly and apply knowledge appropriately.	Has basic understanding of key concepts but explanations may lack depth or clarity. Application of knowledge is somewhat limited.	Shows partial understanding; many concepts are unclear or misunderstood. Struggles to apply knowledge correctly.	Demonstrates little to no understanding of the subject. Unable to explain or apply knowledge.
Performance				
Defines How Well Student Performs				
5	4	3	2	1
Excellent	Good	Satisfactory	Needs Improvement	Poor
Consistently performs tasks with high efficiency, accuracy, and quality. Exceeds expectations.	Perform tasks well with minor errors. Meets expectations consistently.	Perform tasks adequately but with some errors or inconsistencies. Meets basic requirements.	Performance is below expectations with frequent errors and lack of consistency.	Performance is unsatisfactory with major errors and failure to complete tasks properly.
Discipline				
How Well student follows Discipline				
2	1		0	
Excellent	Satisfactory		Poor	
Always punctual, follows rules strictly, and maintains good behavior consistently.	Usually punctual and follow rules, with occasional minor lapses in behavior.		Frequently late, disregards rules, and shows poor behavior.	

2. For Assignments

On time Submission & Completion			
Defines tasks completion and submission			
	1	0	
	On Time	Late/Incomplete	
	Work is submitted fully completed and on or before the deadline.	Work is submitted late or not completed as required.	
Knowledge			
Defines Understanding of a student			
	2	1	0
	Good	Basic	Poor/None
	Demonstrates clear understanding of the topic with accurate and relevant knowledge.	Shows partial understanding; some key points are missing or unclear.	Shows little or no understanding of the topic.
Content and Report			
How Well presented			
	2	1	0
	Good	Basic	Poor/None
	Content is clear, accurate, well-organized, and covers all required points. Reports are neat and easy to follow.	Content covers some points but lacks clarity or detail. Report organization needs improvement.	Content is incomplete, inaccurate, or poorly organized. Reports are unclear or hard to understand.

Term-work Calculations as per Rubric:

Sr. No.	Title	Count	Max Rubric Points	Total Rubric Points	Weight in Marks	Awarded Rubric Points
1	Experiments	11	15	165	15	X (Total marks)
2	Assignments	2	5	10	5	Y (Total marks)
3	Attendance	Attendance Percentage (P)			5	P (Total marks)

$P' = \frac{P*5}{100}$ *(If P=80%, then P' = 4)*

$$TW\ Awarded = (X*15/165)+(Y*5/10)+P`$$

NOTE: All students must note that submission of the work must be completed before the due date of the respective experiments and assignments as mentioned by your teacher. The above rubric is applicable only to the students who submit their completed work before deadline.

Lab Guidelines

1. Attendance & Punctuality: Be present in the lab on time for every session. Maintain attendance by signing in at the start and end of each lab.
2. Lab Preparation: Read and understand the lab exercises before attending the session. Make sure you have the required software (IDE, compiler) installed on your computer.
3. Coding Standards: Write clear, readable code with proper indentation and spacing. Use meaningful variable names and comments to explain your code. Follow standard conventions for braces, loops, and conditionals.
4. Submission Deadlines: Complete and submit your lab exercises by the specified deadlines. Late submissions will be penalized or not accepted unless prior approval is given.
5. Collaboration & Plagiarism: Collaborate with peers for discussions but write your own code. Do not copy- paste solutions from others or online sources. Plagiarism will result in disciplinary action.
6. File Management: Organize your files into folders for each lab. Use appropriate file naming conventions, e.g., lab1_exercise1.c.
7. Backup & Version Control: Maintain backups of your work on cloud storage or a version control system (e.g., Github). Avoid losing work due to computer crashes or file corruption.
8. Documentation: Document your code with comments explaining the purpose of each function and block. Prepare a short report for complex programs if required, explaining the logic, algorithm, and output.
9. Respect Lab Equipment: Handle lab computers and equipment with care. Report any hardware or software issues to the lab assistant immediately.

INDEX

Sr. No.	Title	Page No.	Performed On	Sign & Date	Rubrics Points
1	**Exp 1:** a. WAP to print basic details on screen and format it using escape sequence. b. WAP to get students PCM marks from the user and find the average and eligibility.				
2	**Exp 2:** a. WAP to find if entered number is even or odd. b. WAP to find the sum of all the odd numbers between numbers entered by the user				
3	**Exp 3:** WAP to design a menu driven calculator using switch and goto				
4	**Exp 4:** WAP to find all the prime numbers between two numbers.				
5	**Exp 5:** WAP to find the factorial of a number using iterative and recursive function				
6	**Exp 6:** WAP to define a counter function to print how many times the function is called use storage classes.				
7	**Exp 7:** a. WAP to find the largest element in an array. b. WAP to calculate sum of two matrix.				
8	**Exp 8:** a. WAP to find the length of a string without using library function. b. WAP to check if the entered string is palindrome or not.				
9	**Exp 9:** Design a structure student_record to contain name, roll_number, and total marks obtained. Write a program to read 5 student's data from the user and then display the topper on the screen				
10	**Exp 10:** a. WAP to add two numbers using pointers b. WAP to print the elements of an array in reverse order using pointers				
11	**Exp 11:** WAP to maintain a simple employee database using file handling				
12	Assignment 1				
13	Assignment 2				
14	Attendance				
				TOTAL	

EXPERIMENTS

Expt. No. 1

Title: *WAP to demonstrate Input, Output and Operators.*

Performed On: _____

Sign: _____

Experiment No. 1

CPL (C Programming Lab)

Aim : WAP to demonstrate Input, Output and Operators.

Software : Codeblocks and MingW

Dev Setup : ***Codeblocks + MingW Setup***

Visit the below site to download the setup for the Codeblocks IDE.

https://www.codeblocks.org/downloads/binaries/

Basic Structure of C

```c
#include <stdio.h>

int main()
{


    return 0;
}
```

1. printf and scanf

printf: It is a standard C library function used to output data to the console. The format of printf is:

printf("format string", variable1, variable2, ...);

The format string contains text to be printed and format specifiers that define how the variables should be displayed. Some common format specifiers are:

- %d: for integers
- %f: for floating-point numbers
- %c: for characters
- %s: for strings

Example:

```
int num = 5;
printf("The number is: %d", num);
```

Escape Sequence in C

Escape Sequence	Representation	Description
\\	Backslash (\)	Inserts a single backslash character.
\'	Single Quote (')	Inserts a single quote character.
\"	Double Quote (")	Inserts a double quote character.
\n	Newline	Moves the cursor to the next line.
\t	Horizontal Tab	Inserts a horizontal tab space.
\r	Carriage Return	Moves the cursor to the beginning of
\a	Alert (Bell)	Triggers an alert sound (beep).
\b	Backspace	Moves the cursor one position back.

scanf: It is used to read input from the user. It takes input from the standard input (usually the keyboard) and stores it in the provided variables. Its syntax is:

scanf("format string", &variable1, &variable2, ...);

The format specifiers used in scanf are the same as in printf. Variables passed to scanf must have their memory addresses passed using the & operator.

Example:

```
int num;

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

2. Operators in C

C provides various operators for performing operations on data. These operators are divided into categories:

- **Arithmetic Operators**: Used for mathematical calculations.
  - + (Addition)
  - - (Subtraction)
  - * (Multiplication)
  - / (Division)
  - % (Modulus)
- **Relational Operators**: Used to compare values.
  - == (Equal to)

  - != (Not equal to)
  - > (Greater than)
  - < (Less than)
  - >= (Greater than or equal to)
  - <= (Less than or equal to)
- **Logical Operators**: Used to perform logical operations.
  - && (Logical AND)
  - || (Logical OR)
  - ! (Logical NOT)
- **Assignment Operators**: Used to assign values to variables.
  - = (Assign)
  - +=, -=, *=, /=, %= (Compound assignment)
- **Increment/Decrement Operators**: Used to increase or decrease a value by one.
  - ++ (Increment)
  - -- (Decrement)
- **Bitwise Operators**: Used to perform bit-level operations.
  - & (AND)
  - | (OR)
  - ^ (XOR)
  - ~ (NOT)
  - << (Left shift)
  - >> (Right shift)

Task 1 : **WAP to print your name, age, class, division and UIN on the screen. Use escape sequences to properly format the output.**

Program with Output:

Pre-Lab Questions:

1. What are escape sequences in C and why are they used?
2. How can you format the output in C using escape sequences?
3. What are some common escape sequences used in C programming?

Post-Lab Questions:

1. What escape sequences did you use in the program to format the output?
2. How would you print a backslash (\) in your program using escape sequences?
3. How would you modify the program to include more fields, such as phone number, using proper formatting?

Task 2: **WAP to get students PCM marks from the user and find the average. Use conditional operator to print if the student is eligible for admission. Eligibility criteria is 50% in PCM.**

Program with Output:

Pre-Lab Questions:

1. What data types will you use to store the marks in this program?
2. What formula will you use to calculate the average of three numbers?
3. How will you implement conditional logic to check eligibility in this program?

Post-Lab Questions:

1. What was the output of the program when you tested it with different marks?
2. What would happen if you input negative marks? How can you handle such cases in the code?
3. How can you modify the program to get input for more subjects or add more conditions for eligib

Conclusion:

……

……

CO's Covered:

……

……

Expt. No. 2

Title: *WAP to find the sum of odd numbers between two numbers entered by the user.*

Performed On: _____

Sign:_____

Experiment No. 2

CPL (C Programming Lab)

Aim : WAP to find the sum of odd numbers between two numbers entered by the user.

Software : Codeblocks & MingW

Theory : ***Control Structures***

1. if-else Statements

The if-else construct is used for decision-making in C programming. It allows the program to execute certain code blocks conditionally based on whether a specified condition is true or false.

- **if statement**: Executes a block of code if a given condition is true.

```
if (condition) {
    // Code to execute if condition is true
}
```

- **else statement**: Executes a block of code if the condition in the if statement is false.

```
if (condition) {
    // Code if condition is true
} else {
    // Code if condition is false
}
```

- **else if ladder**: Used when multiple conditions need to be checked.

```
if (condition1) {
    // Code if condition1 is true
} else if (condition2) {
    // Code if condition2 is true
} else {
    // Code if none of the conditions are true
```

```
}
```

Example:

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

2. Loops

Loops allow the repetition of a block of code multiple times, based on a condition. C has three types of loops:

a. for Loop:

Used when the number of iterations is known. It contains three parts: initialization, condition, and increment/decrement.

```
for (initialization; condition; increment/decrement) {
    // Code to be repeated
}
```

Example:

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

b. while Loop:

Executes the code block as long as the condition remains true. It checks the condition before each iteration.

```
while (condition) {
    // Code to be repeated
```

```
}
```

Example:

```
int i = 0;
while (i < 5) {
    printf("%d ", i);
    i++;
}
```

c. do-while Loop:

Similar to the while loop but the condition is checked after executing the loop body, so it runs at least once.

```
do {
    // Code to be repeated
} while (condition);
```

Example:

```
int i = 0;
do {
    printf("%d ", i);
    i++;
} while (i < 5);
```

Post-Lab Questions:

1. **What changes would you make if the program needed to find the sum of even numbers instead of odd?**
   How can you modify the condition inside the loop to sum even numbers?
2. **How would you handle cases where the starting number is larger than the ending number?**
   What changes can be made to ensure the program works regardless of the order of input?
3. **What was the output when the starting and ending numbers were the same?**
   How does the program behave in this case, and is it expected?
4. **How can you modify the program to avoid counting negative numbers if they are entered?**
   If the user enters negative numbers, how can you ensure that only positive odd numbers are considered in the sum?

Task 1: **WAP to find if entered number is even or odd. (Draw flowchart also)**

Program with Output:

Task 2 : **WAP to find the sum of all the odd numbers between numbers entered by the user. (Draw flowchart also)**

Program with Output:

Conclusion:

……

……

CO's Covered:

……

……

Expt. No. 3

Title: *WAP to design a menu driven calculator using switch and goto.*

Performed On: _____

Sign:_____

Experiment No. 3

CPL (C Programming Lab)

Aim : WAP to design a menu driven calculator using switch and goto.

Software : Codeblocks & MingW

Theory : ***Switch and Goto Statements***

1. switch Statement

The switch statement is a control structure used for multi-way branching. It allows a variable or expression to be tested against a list of values, and the code corresponding to the matching value is executed. It is often used when there are multiple conditions to evaluate, typically as an alternative to else if ladders.

Syntax:

```
switch (expression) {
    case constant1:
        // Code for case 1
        break;
    case constant2:
        // Code for case 2
        break;
    ...
    default:
        // Code if no case matches
}
```

- **Expression**: The value (usually an integer or character) that is compared to the constants in the case statements.
- **case**: Each case defines a possible value of the expression.
- **break**: Used to exit the switch after a matching case is executed, preventing "fall-through" to subsequent cases.

- **default**: An optional block that executes if no case matches.

Example:

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

2. goto Statement

The goto statement provides a way to jump to a labeled section of code. While it offers flexibility in controlling the flow of execution, its use is generally discouraged because it can make code difficult to follow and maintain (often referred to as "spaghetti code").

Syntax:

```
goto label;
// Some code
label:
   // Code to jump to
```

- **goto**: This keyword is followed by the name of a label, and execution jumps to that label.
- **Label**: A user-defined identifier followed by a colon (:) that marks the destination of the goto statement.

Example:

```
int num = 1;
if (num == 1)
   goto label;
printf("This won't be printed");
label:
printf("Jumped to label");
```

Task 1: **WAP to design a menu driven calculator using switch and goto.**

Program with Output:

Conclusion:

……

……

CO's Covered:

……

……

Expt. No. 4

Title: *WAP to find all the prime numbers between two numbers.*

Performed On: _____

Sign:_____

Experiment No. 4

CPL (C Programming Lab)

Aim : WAP to find all the prime numbers between two numbers using functions.

Software : Codeblocks & MingW

Theory : ***Functions***

A **function** in C is a block of code that performs a specific task. Functions help organize and reuse code, making programs more modular, easier to understand, and maintain.

Key Concepts of Functions:

1. **Function Declaration (Prototype)**: The function must be declared before it is used. A function declaration specifies the function's name, return type, and parameters.

return_type function_name(parameter_type1 param1, parameter_type2 param2, ...);

Example:

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

2. **Function Definition**: This is where the actual logic of the function is implemented. It includes the function header (same as the declaration) and the body, which contains the statements to be executed.

```
return_type function_name(parameter_type1 param1, parameter_type2 param2, ...) {
   // Function body (code to perform the task)
   return value;  // If the function has a return type other than `void`
}
```

Example:

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

3. **Function Call**: A function is called or invoked by writing its name followed by parentheses containing arguments, if any. The arguments passed to the function must match the parameters in the function definition in type and number.

```
function_name(argument1, argument2, ...);
```

Example:

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

Types of Functions in C:

1. **Standard Library Functions**: Functions that are provided by C's standard library, such as printf(), scanf(), sqrt(), etc.
2. **User-Defined Functions**: Functions created by the programmer to perform specific tasks.

Key Components:

- **Return Type**: Specifies the type of value the function will return. It can be any valid C data type like int, float, char, or void (if no value is returned).
- **Function Name**: Identifier by which the function is called. It should follow C's naming rules.
- **Parameters (Arguments)**: Inputs to the function. The parameters are specified within the parentheses. If no parameters are needed, the parentheses are left empty.
- **Return Statement**: If the function has a return type other than void, it must include a return statement to send a value back to the calling function.

Example of a Function:

```
#include <stdio.h>

// Function declaration

int multiply(int x, int y);


int main() {

   int result = multiply(5, 3);  // Function call

   printf("Result: %d", result);

   return 0;

}


// Function definition
```

```
int multiply(int x, int y) {
    return x * y;
}
```

Advantages of Using Functions:

- **Modularity**: Functions break the program into smaller, manageable sections.
- **Reusability**: Once defined, a function can be used (called) multiple times in the program.
- **Maintainability**: Functions make code easier to maintain and debug.
- **Code Readability**: Functions enhance the structure and clarity of the code.

Post-Lab Questions:

1. How can you optimize the prime-checking function to make it faster?
2. How did you handle edge cases (e.g., when both numbers are the same or when the range includes negative numbers)?

Task 1: **WAP to find all the prime numbers between two numbers using functions. (Draw flowchart also)**

Program with Output:

Conclusion:

……

……

CO's Covered:

……

……

Expt. No. 5

Title: *WAP to find the factorial of a number using iterative and recursive function.*

Performed On: _____

Sign:_____

Experiment No. 5

CPL (C Programming Lab)

Aim : WAP to find the factorial of a number using iterative and recursive function.

Software : Codeblocks & MingW

Theory : ***Iterative and Recursive Functions***

1. Iterative Functions

An **iterative function** uses loops (like for, while, or do-while) to repeatedly execute a block of code until a specific condition is met.

Key Characteristics:

- Utilizes looping constructs for repetition.
- Generally more efficient in terms of memory since it avoids the overhead of multiple function calls.
- Easier to understand and debug for basic problems.

Example: Iterative Function to Calculate Factorial

```
int factorial_iterative(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```

Advantages of Iterative Functions:

- Lower memory usage (no extra function calls or stack frames).
- Efficient for problems requiring many repetitions.

Disadvantages:

- Some problems are more naturally expressed in recursive terms (e.g., tree traversals).

2. Recursive Functions

A **recursive function** is a function that calls itself, directly or indirectly, to solve a smaller instance of the same problem. It typically includes a **base case** (which stops the recursion) and a **recursive case** (where the function calls itself).

Key Characteristics:

- Breaks down the problem into smaller sub-problems.
- Requires a base case to terminate, avoiding infinite recursion.
- Uses the call stack for each recursive call, which can lead to higher memory usage.

Example: Recursive Function to Calculate Factorial

```
int factorial_recursive(int n) {
    if (n == 0 || n == 1)  // Base case
        return 1;
    else
        return n * factorial_recursive(n - 1);  // Recursive case
}
```

Advantages of Recursive Functions:

- Suitable for problems that naturally follow a recursive pattern (e.g., tree traversal, mathematical sequences).
- Leads to cleaner and simpler code for problems that can be divided into sub-problems.

Disadvantages:

- Higher memory usage due to function calls stacking up in the call stack.
- Can lead to stack overflow if recursion depth is too high.

Post Lab Questions

1. Which method is better iterative or recursive for large numbers

Task 1 & 2: **WAP to find the factorial of a number using iterative and recursive functions.**

Program with Output:

Conclusion:

……

……

CO's Covered:

……

……

Expt. No. 6

Title: *WAP to define a counter function to print how many times the function is called. use storage classes.*

Performed On: _____

Sign:_____

Experiment No. 6

CPL (C Programming Lab)

Aim : WAP to define a counter function to print how many times the function is called. use storage classes.

Software : Codeblocks & MingW

Theory : ***Storage Classes***

1. if-else Statements

Storage classes in C define the **scope, visibility, lifetime**, and **memory location** of variables or functions. They determine how variables are stored, how long they stay in memory, and how they are accessed within a program.

The four primary storage classes in C are:

1. **auto**
2. **register**
3. **static**
4. **extern**

1. auto (Automatic Storage Class)

- **Default storage class** for local variables declared inside functions or blocks.
- **Scope**: Local to the function/block where it is declared.
- **Lifetime**: Exists only during the execution of the block/function. It is destroyed when the block/function ends.
- **Storage**: Stored in memory (RAM).
- **Keyword**: auto (though it's optional since variables are auto by default).

Example:

```
void function() {
   auto int x = 10;  // or just 'int x = 10;'
}
```

2. register (Register Storage Class)

- Used to store variables in the **CPU register** for faster access, instead of RAM. Recommended for variables that are frequently used.
- **Scope**: Local to the function/block.
- **Lifetime**: Same as auto (exists during function/block execution).
- **Storage**: Stored in the CPU register (if available). If not, it is stored in RAM.
- **Keyword**: register.
- **Limitations**: Cannot take the address of a register variable (i.e., no pointers).

Example:

```
void function() {
   register int count = 0;  // Suggest storing in a CPU register
}
```

3. static (Static Storage Class)

- **Scope**: Local to the function/block if declared inside, but retains its value between function calls.
- **Lifetime**: Exists for the entire lifetime of the program (persistent).
- **Global static**: Limits the scope of a global variable or function to the file in which it is declared.
- **Keyword**: static.
- **Use Case**: Useful for keeping state in a function across multiple calls.

Example:

```
void function() {
   static int count = 0;  // Value is retained between calls
   count++;
   printf("%d", count);
}
```

4. extern (External Storage Class)

- Used to declare a **global variable or function** that is accessible across multiple files.

- **Scope**: Global (if declared outside all functions).
- **Lifetime**: Exists for the entire program execution.
- **Keyword**: extern.
- **Use Case**: When a variable or function is shared across multiple files. Declared with extern in other files, but defined once (without extern).

Example:

```
// In file1.c
int x = 10;  // Definition

// In file2.c
extern int x;  // Declaration (refers to the variable defined in file1)
```

Task 1: **WAP to define a counter function to print how many times the function is called. use storage classes.**

Program with Output:

Conclusion:

……

……

CO's Covered:

……

……

Expt. No. 7

Title: *WAP to find the second largest element in an Array & WAP to calculate sum of two matrix.*

Performed On: _____

Sign:_____

Experiment No. 7

CPL (C Programming Lab)

Aim : WAP to find the largest element in an Array & WAP to calculate sum of two matrix.

Software : Codeblocks & MingW

Theory : ***Arrays***

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 and access them using indices.

Key Concepts:

1. **Declaration of Arrays**: Arrays must be declared with a specific size and data type. The size determines how many elements the array can hold.

   data_type array_name[size];

   Example:

   int numbers[5]; // Declares an integer array of size 5

2. **Initialization of Arrays**: Arrays can be initialized at the time of declaration. If fewer elements are provided during initialization, the remaining elements are automatically initialized to zero.

   int numbers[5] = {1, 2, 3, 4, 5}; // Fully initialized

   int numbers[5] = {1, 2}; // Partially initialized, remaining elements are 0

3. **Accessing Array Elements**: Array elements are accessed using their index. The index starts at 0 and goes up to size-1.

   array_name[index];

   Example:

   numbers[0] = 10; // Assigns value to the first element

   printf("%d", numbers[0]); // Accesses and prints the first element

4. **Types of Arrays**:

   - **One-dimensional array**: A simple list of elements. Example:

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

- **Multi-dimensional arrays**: Arrays that contain more than one dimension (e.g., 2D arrays like matrices). Example (2D array):

```
int matrix[3][3] = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};
```

Advantages of Arrays:

- **Efficiency**: Arrays store data in contiguous memory locations, allowing fast access to elements using indices.
- **Organized storage**: You can manage multiple variables of the same type under one name.
- **Easy iteration**: Arrays can be easily traversed using loops.

Disadvantages of Arrays:

- **Fixed size**: Once declared, the size of an array cannot be changed dynamically.
- **Homogeneous data**: Arrays can only store elements of the same data type.

Task 1: **WAP to find the largest element in an array.**

Program with Output:

Task 2 : **WAP to calculate sum of two matrix.**

Program with Output:

Conclusion:

……

……

CO's Covered:

……

……

Expt. No. 8

Title: *WAP to find the length of a string without using library function & WAP to check if the entered string is palindrome or not.*

Performed On: _____

Sign:_____

Experiment No. 8

CPL (C Programming Lab)

Aim : WAP to find the length of a string without using library function & WAP to check if the entered string is palindrome or not.

Software : Codeblocks & MingW

Theory : ***Strings***

In C, **strings** are defined as arrays of characters, terminated by a special character called the **null character** (\0). This null character signals the end of the string, distinguishing it from a simple character array.

Key Concepts:

1. **Declaration of Strings**: A string is declared as a character array. The size of the array must be sufficient to hold the characters and the null character (\0).

   char str[size];

   Example:

   char name[10]; // Declares a string that can hold up to 9 characters + '\0'

2. **Initialization of Strings**: Strings can be initialized at the time of declaration.
   - **Method 1**: Assign individual characters, ensuring the last character is \0.

     char name[5] = {'H', 'e', 'l', 'l', 'o'};

   - **Method 2**: Initialize using a string literal, which automatically adds the null terminator.

     char name[] = "Hello"; // 'H', 'e', 'l', 'l', 'o', '\0'

3. **Accessing String Elements**: Individual characters in a string can be accessed using array indexing.

   char first_letter = str[0]; // Accesses the first character

4. **Common String Functions**: The C standard library provides several functions for handling strings, defined in the string.h header file:
   - **strlen()**: Returns the length of the string (excluding the null character).

     int len = strlen(str);

- **strcpy()**: Copies one string to another.

  strcpy(destination, source);

- **strcat()**: Concatenates (joins) two strings.

  strcat(destination, source);

- **strcmp()**: Compares two strings.

  int result = strcmp(str1, str2);

String Handling Challenges in C:

- **Fixed size**: Once the size of the array is declared, it cannot be changed dynamically. This can limit flexibility.
- **Manual management**: C does not provide native string types like modern languages. You must manage memory and handle strings manually using arrays and functions.
- **Null termination**: The null character (\0) must always be present at the end of the string, or string operations may cause undefined behavior.

Task 1: **WAP to find the length of a string without using library function.**

Program with Output:

Task 2 : **WAP to check if the entered string is palindrome or not.**

Program with Output:

Conclusion:

……

……

CO's Covered:

……

……

Expt. No. 9

Title: *Design a structure student_record to contain name, roll_number, and total marks obtained. Write a program to read 5 students data from the user and then display the topper on the screen.*

Performed On: _____

Sign:_____

Experiment No. 9

CPL (C Programming Lab)

Aim : Design a structure student_record to contain name, roll_number, and total marks obtained. Write a program to read 5 students data from the user and then display the topper on the screen.

Software : Codeblocks & MingW

Theory : ***Structures***

1. if-else Statements

A **structure** in C (also known as struct) is a user-defined data type that allows grouping of variables of different data types under a single name. Structures are useful for organizing complex data and represent entities with multiple attributes.

Key Concepts:

1. **Structure Declaration**: A structure is declared using the struct keyword, followed by the structure name and the variables (members) it holds within curly braces {}.

```
struct structure_name {
   data_type  member1;
   data_type member2;
   ...
};
```

Example:

```
struct Person {
   char name[50];
   int age;
   float height;
};
```

2. **Structure Definition and Initialization**: After declaring a structure, you can define variables of that structure type and optionally initialize them.

```
struct Person person1 = {"Alice", 25, 5.6};  // Initialization
```

3. **Accessing Structure Members**: Structure members are accessed using the dot (.) operator.

   variable_name.member_name

   Example:

   printf("%s is %d years old.", person1.name, person1.age);

4. **Nested Structures**: Structures can contain other structures as members, allowing for more complex data representation.

   Example:

```
struct Address {
   char city[50];
   int zip_code;
};

struct Person {
   char name[50];
   struct Address addr;
};
```

5. **Passing Structures to Functions**: Structures can be passed to functions by value or by reference (using pointers). When passed by value, a copy of the structure is made. When passed by reference, the original structure is modified.

   Example:

```
void display(struct Person p) {
   printf("Name: %s, Age: %d", p.name, p.age);
}
```

Task 1: **Design a structure student_record to contain name, roll_number, and total marks obtained. Write a program to read 5 students data from the user and then display the topper on the screen.**

Program with Output:

Conclusion:

……

……

CO's Covered:

……

……

Expt. No. 10

Title: *WAP to add two numbers using pointers & WAP to print the elements of an array in reverse order using pointers.*

Performed On: _____

Sign:_____

Experiment No. 10

CPL (C Programming Lab)

Aim : WAP to add two numbers using pointers & WAP to print the elements of an array in reverse order using pointers..

Software : Codeblocks & MingW

Theory : ***Pointers***

A **pointer** in C is a variable that stores the **memory address** of another variable. Pointers provide a powerful way to manipulate memory directly, allowing for efficient array handling, dynamic memory allocation, and the ability to pass variables by reference to functions.

Key Concepts:

1. **Declaration of Pointers**: Pointers are declared using the asterisk (*) symbol. The data type of the pointer must match the data type of the variable it will point to.

   *data_type *pointer_name;*

   Example:

   *int *ptr; // Declares a pointer to an integer*

2. **Initialization of Pointers**: A pointer is initialized by assigning it the address of a variable, which is done using the address-of operator (&).

   int x = 10;

   *int *ptr = &x; // ptr holds the address of x*

3. **Dereferencing a Pointer**: Dereferencing a pointer means accessing the value stored at the memory address that the pointer holds. The dereference operator (*) is used for this purpose.

   *int value = *ptr; // Retrieves the value stored at the address in ptr*

4. **Pointer Arithmetic**: Since pointers store addresses, they can be incremented or decremented to move between memory locations. For example, incrementing a pointer to an int moves it to the next integer position in memory.

   ptr++; // Moves the pointer to the next integer in memory

5. **Null Pointers**: A pointer can be assigned a special value NULL to indicate that it points to nothing.

```
int *ptr = NULL;  // Points to nothing
```

6. **Pointers and Arrays**: Arrays and pointers are closely related. The name of an array is a pointer to its first element, and you can use pointers to traverse through array elements.

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

int *ptr = arr;  // Points to the first element of the array
```

7. **Pointers to Pointers**: A pointer can point to another pointer, creating multiple levels of indirection.

```
int x = 10;

int *ptr = &x;

int **ptr_to_ptr = &ptr;  // Pointer to a pointer
```

Example of Pointer Usage:

```
#include <stdio.h>


int main() {
   int x = 10;
   int *ptr = &x;  // Pointer to x


   printf("Address of x: %p\n", ptr);  // Prints the address of x
   printf("Value of x: %d\n", *ptr);     // Dereferences the pointer to get the value of x


   return 0;
}
```

Key Operators:

- **& (Address-of operator)**: Used to get the memory address of a variable.

```
int *ptr = &x;  // Address of x is assigned to ptr
```

- *** (Dereference operator)**: Used to access the value at the memory address stored in the pointer.

*int value = *ptr; // Gets the value at the address ptr is pointing to*

Advantages of Pointers:

- **Efficient Memory Access**: Pointers allow direct access to memory, which can improve performance for certain operations.
- **Dynamic Memory Allocation**: Pointers enable the allocation of memory at runtime using functions like malloc(), calloc(), and free().
- **Pass-by-Reference**: Pointers allow functions to modify variables by reference, without copying the data.

Task 1: **WAP to add two numbers using pointers.**

Program with Output:

Task 2 : **WAP to print the elements of an array in reverse order using pointers.**

Program with Output:

Conclusion:

……

……

CO's Covered:

……

……

Expt. No. 11

Title: *WAP to maintain a simple employee database using file handling.*

Performed On: _____

Sign:_____

Experiment No. 11

CPL (C Programming Lab)

Aim : WAP to maintain a simple employee database using file handling.

Software : Codeblocks & MingW

Theory : ***File Handling***

File handling in C enables programs to **create, read, write**, and **modify** files stored on disk. It provides a way to store data permanently beyond the lifetime of a program's execution. C uses standard library functions for file operations, defined in the stdio.h header.

Key Concepts:

1. **File Pointer**: In C, a **file pointer** is used to access and manipulate files. It is declared as:

   *FILE *file_pointer;*

2. **Opening a File**: Files are opened using the fopen() function, which requires the file name and the mode of operation (read, write, etc.). The function returns a pointer to the file, or NULL if the file cannot be opened.

   *FILE *fopen(const char *filename, const char *mode);*

   Modes for opening files:

   - "r": Open a file for **reading**. The file must exist.
   - "w": Open a file for **writing**. If the file exists, it is truncated (emptied); otherwise, a new file is created.
   - "a": Open a file for **appending**. Data is written at the end of the file.
   - "r+": Open a file for **both reading and writing**. The file must exist.
   - "w+": Open a file for **reading and writing**. It overwrites existing content or creates a new file.
   - "a+": Open a file for **reading and appending**. Data is added to the end, but previous data is not modified.

   Example:

   ```
   FILE *file = fopen("example.txt", "r");
   if (file == NULL) {
   ```

```
    printf("File not found.");
}
```

3. **Closing a File**: After completing file operations, you must close the file using fclose() to release the file pointer and ensure data is written to disk.

```
fclose(file_pointer);
```

4. **Reading from a File**: C provides several functions to read data from a file:
   - fgetc(): Reads a single character from the file.

```
char c = fgetc(file_pointer);
```

   - fgets(): Reads a string from the file.

```
fgets(buffer, size, file_pointer);
```

   - fscanf(): Reads formatted input from the file (similar to scanf).

```
fscanf(file_pointer, "%d", &variable);
```

5. **Writing to a File**: Similar to reading, C provides functions for writing data to files:
   - fputc(): Writes a single character to the file.

```
fputc('A', file_pointer);
```

   - fputs(): Writes a string to the file.

```
fputs("Hello, World!", file_pointer);
```

   - fprintf(): Writes formatted output to the file (similar to printf).

```
fprintf(file_pointer, "Name: %s, Age: %d", name, age);
```

6. **Binary File Handling**: In addition to text files, C can handle **binary files** for reading and writing raw data (e.g., images, executables). Use modes "rb" (read binary) and "wb" (write binary) for binary file operations.

   Example:

```
FILE *file = fopen("data.bin", "wb");
int data = 100;
fwrite(&data, sizeof(int), 1, file);  // Writing binary data
fclose(file);
```

7. **Error Handling**: It's important to check whether file operations succeed, particularly when opening or reading files. Functions like fopen() return NULL if the operation fails.

Example of File Handling in C:

```
#include <stdio.h>

int main() {
    // Opening a file for writing
    FILE *file = fopen("example.txt", "w");
    if (file == NULL) {
        printf("Error opening file.\n");
        return 1;
    }

    // Writing to the file
    fprintf(file, "Hello, file handling in C!\n");

    // Closing the file
    fclose(file);

    // Opening the file for reading
    file = fopen("example.txt", "r");
    if (file == NULL) {
        printf("Error opening file.\n");
        return 1;
    }

    // Reading from the file
    char buffer[100];
    fgets(buffer, 100, file);
    printf("File content: %s", buffer);
    // Closing the file
    fclose(file);
    return 0;
}
```

Task 1: **WAP to maintain a simple employee database using file handling.**

Program with Output:

Conclusion:

……..

……..

CO's Covered:

……..

……..


Keep
Practicing

Syllabus

Theory Syllabus:

Sr. No.	Module	Detailed Content	Hours	LO Mapping
1	**Fundamentals of C-Programming**	1.1 Character Set, Identifiers and keywords, Data types, Constants, Variables. 1.2 **Operators**-Arithmetic, Relational and logical, Assignment, Unary, Conditional, Bitwise, Comma, other operators. Expression, statements, Library Functions, Preprocessor. 1.3 **Data Input and Output** getchar( ), putchar( ), scanf( ), printf( ), gets( ), puts( ), Structure of C program .	**06**	**LO1, LO2**
2	**Control Structures**	**2.1 Branching** - If statement, If-else Statement, Multiway decision. **2.2 Looping** – while, do-while, for **2.3 Nested control structure**- Switch statement, Continue statement Break statement, Goto statement.	**05**	**LO3**
3	**Functions and Parameter**	**3.1** Function -Introduction of Function, Function Main, defining a Function, accessing a Function, Function Prototype, Passing Arguments to a Function, Recursion. **3.2** Storage Classes Auto , Extern , Static, Register	**05**	**LO3**
4	**Arrays , String Structure**	**4.1 Array**-Concepts, Declaration, Definition, Accessing array element, One-dimensional and Multidimensional array. **4.2 String**- Basic of String, Array of String, Functions in String.h **4.3 Structure**- Declaration, Initialization, structure within structure, Operation on structures, Array of Structure.	**05**	**LO4**
5	**Pointer**	**5.1 Pointer:** Introduction, Definition and uses of Pointers, Address Operator, Pointer Variables, Dereferencing Pointer, Void Pointer, Pointer Arithmetic, Pointers to Pointers, Pointers and Array.	**03**	**LO5**
6	**Files**	**6.1 File**s: File operation- Opening, Closing, Creating, Reading, Processing File.	**02**	**LO6**

Practicals Syllabus:

Sr No	Suggested List of Experiments	Hrs
01	a) Program to demonstrate Operators Data Input and Output – getchar( ), putchar( ), scanf( ), printf( ), gets( ), puts( ) b) Program to demonstrate Operators-Arithmetic, Relational and logical, Assignment, Unary, Conditional, Bitwise, Comma, other operators.	**02**
02	a) Program to demonstrate Branching - If statement, If-else Statement, Multiway decision. b) Program to demonstrate Looping – while, do-while	**02**
03	a) Program to demonstrate Nested control structure- Switch statement, Continue statement, Break statement, Goto statement	**02**
04	a) Program to demonstrate Function, Passing Arguments to a Function (call by value and call by reference	**02**
05	a) Implement an iterative function for factorial/ Fibonacci etc. b) Implement a recursive function for factorial/ Fibonacci etc.	**02**
06	Program to demonstrate Storage Classes Auto, Extern, Static, Register	**02**
07	a. Program to demonstrate Array 1D, b. Program to demonstrate Array 2D	**02**
08	a. Program to demonstrate String b. Program to demonstrate String arrays of string	**02**
09	Program to demonstrate Structure : Write a program to store and display information of a student/employee etc. using structures. a) Define a structure. b) Read and store details. c) Display the stored information.	**02**
10	Program to demonstrate pointers a) Define a node structure. Implement functions to insert, delete, and display nodes.	**02**
11	Program to demonstrate files Write a program to maintain a simple student/employee etc. database using file handling. a) Open a file to store student records. b) Implement functions to add, update, and display records. Ensure data persistence by saving changes to the file.	**02**
12	Implement one small application using Function, Files, Structure and Pointers concepts you have learnt in C (eg. : Simple Library Management System 1.Functions: Add, display, and search books. 2. Files: Store and retrieve book data. 3. Structures: Represent a book. 4. Pointers: Manage the list of books dynamically	**02**