VSEC102

RIZVI EDUCATION SOCIETY'S

Rizvi College of Engineering

Department of Humanities & Sciences

C Programming Lab

NAME:

CLASS & Div:

ROLL NO. / UIN:

Stick QR Code Here from CMsys

RIZVI EDUCATION SOCIETY'S

Rizvi College of Engineering

Department of Humanities & Sciences

Lab File

Class & Semester: *FE (Sem 1)*

Division: ... ...

Subject: *CProgramming Lab*

Name: ……

Roll No.:……..

UIN: ……

Document Revised on: *08 / 09 / 2024*

RIZVI EDUCATION SOCIETY'S

Rizvi College of Engineering

Department of Humanities & Sciences

Certificate

This is to certify that ..... of the First-year Humanities and Sciences Department, Division ..... has performed all the experiments of C Programming Lab with satisfactory results during the academic year 2024-25.

....

Name & Sign *Name & Sign*

Subject In-charge

H.o.D. Department of Humanities & Sciences

....

Name & Sign *Name & Sign*

Internal Examiner **External Examiner**

INSTITUTE VISION

To become a leading entity in transforming the diverse class of learners into innovators, analyzers and entrepreneurs competent to develop eco-friendly sustainable solutions and work in multi-disciplinary environment to meet the global challenges and contribute towards nation building.

INSTITUTE MISSION

Impart Core Fundamental Principles: To impart core fundamental principles of engineering and science, through conventional and innovative methods which will help the learners acquire skillset to create solutions to complex engineering problems..

Bridge the Technical Skill Gap: To bridge the industry – academia gap through curriculum enrichment activities for industry readiness.

Inculcate Professional Etiquettes and Ethics: To groom the learners through well-planned training & placement activities laced with professional etiquettes and ethics leading to their holistic development thus enabling them to acquire distinguished positions in prominent organizations and inculcating lifelong learning capability.

ResearchandDevelopment: To provide modern infrastructure and the necessary resources, for planning and implementing innovative research ideas, leading to entrepreneurship.

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

Code Functionality				
Workingofthecodeas per specs				
	3	2	1	0
	Excellent	Good	Average	Poor
	Program meets all requirements and produces correct output with no errors or bugs.	Program meets most requirements with minor errors or issues.	Program meets some requirements but has errors affecting functionality.	Program does not meet requirements and has major functionality issues.
Code Readability & Structure				
Proper documentation and naming convention and indentation				
	3	2	1	0
	Excellent	Good	Average	Poor
	Code is well-organized with proper indentation, meaningful variable names, and comments	Code is organized but lacks sufficient comments or has minor readability issues.	Code is difficult to read, with poor structure and missing comments.	Code is unreadable with no structure or comments.
Plagiarism				
AmountofPlagiarism				
	3	2	1	0
	Excellent	Good	Average	Poor
	Unique and Innovative Work	Extremely low plagiarism	Some amount of Plagiarism / duplicate work found	High plagiarism found
Outputs				
Properoutputs attached				
	3	2	1	0
	Excellent	Good	Average	Poor
	Proper output is attached with good formatting of the contents	Proper output but lacks good formatting	Some of the output is inappropriate	Incorrect output
Source Code Versioning				
Use of any source code versioning tools like github / bitbucket etc.				
	1		0	
	Yes		No	
	The code is committed to version control and appropriate screenshot attached		No version control used	

2. For Assignments

Understanding of Concepts			
Howwellthestudentunderstood the concept			
	2	1	0
	Good	Average	Poor
	Demonstrates a thorough understanding of C programming concepts. Answers are detailed and accurate.	Shows basic understanding with some significant inaccuracies.	Shows little to no understanding of the concepts.
Correctness of Answers			
How appropriate and correct are the answers			
	2	1	0
	Good	Average	Poor
	All answers are correct and thoroughly justified with examples or reasoning.	Some answers are correct, but several errors or incomplete justifications are present.	Answers are mostly incorrect or lacking justification.
Plagiarism			
ContentOriginality			
	2	1	0
	Good	Average	Poor
	Assignment is written in own words without any plagiarism	Some amount of content is plagiarized	Most of the content plagiarized

Termwork Calculations as per Rubric:

Sr. No.	Title	Count 11	Max Rubric Points	Total Rubric Points	Weightage in Marks	Awarded Rubric Points
1	Experiments	2	13	143	15	X
2	Assignments		6	12	5	Y
3	Attendance	Attendance Percentage (P)			5	P'

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

$$TW\ Awarded = \frac{X * 15}{143} + \frac{Y * 5}{12} + P'$$

NOTE: (*) For Group Project Students can form a group of 3 to 5 students to complete any complex project in c language. This will help students to develop advanced skill set and team work skills.

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 rubrics 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:** WAPtodesignamenudriven calculator usingswitchandgoto				
4	**Exp 4:** WAPtofindalltheprimenumbers between twonumbers.				
5	**Exp 5:** WAPtofindthefactorialofanumber using iterativeandrecursivefunction				
6	**Exp 6:** WAPtodefineacounterfunctionto print howmanytimesthefunctioniscalled use storageclasses.				
7	**Exp 7:** a. WAPtofindthelargestelementinan array. b. WAPtocalculatesumoftwomatrix.				
8	**Exp 8:** a. WAPtofindthelengthofastring without using library function. b. WAPtocheckiftheenteredstringis palindrome or not.				
9	**Exp 9:** Designastructurestudent_recordto contain name,roll_number,andtotalmarks obtained. Writeaprogramtoread5students data from theuserandthendisplaythetopper on the screen **Exp 10:**				
10	a. WAP to addtwonumbersusingpointers b. WAP to printtheelementsofanarrayin reverse orderusingpointers **Exp 11:** WAPtomaintainasimpleemployee				
11	database usingfilehandling Assignment 1 Assignment 2				
12	Attendance				
13					
14		--			
				TOTAL	

EXPERIMENTS

Expt. No. 1

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

#	Rubrics	Points
1	Code Functionality (0 – 3)	
2	Code Readability & Structure (0 – 3)	
3	Plagiarism (0 – 3)	
4	Outputs (0 – 3)	
5	Source Code Versioning (0 – 1)	
	Total Points	

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

```
#include <stdio.h>

int main()
{


    return 0;
}
```

1. printf and scanf

printf: It isastandard 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:

```
intnum= 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:

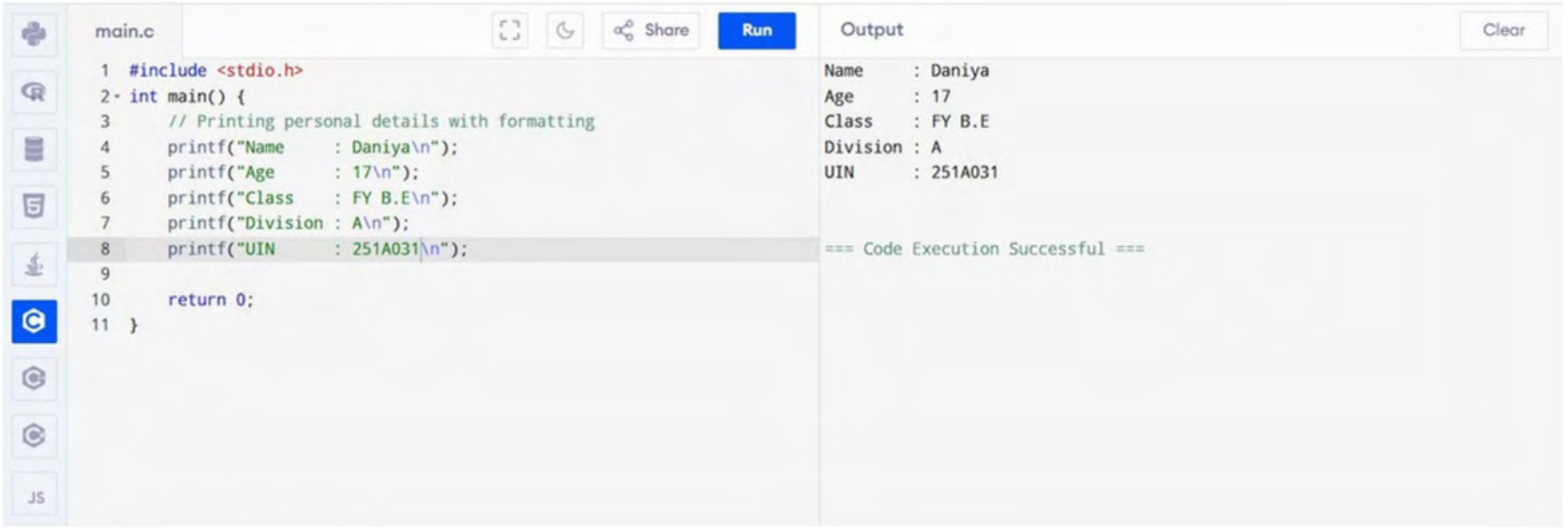
- **Arithmetic Operators**: Used for mathematical calculations.
  - o + (Addition)
  - o - (Subtraction)
  - o * (Multiplication)
  - o / (Division)
  - o % (Modulus)
- **Relational Operators**: Used to compare values.
  - o == (Equal to)

- != (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?

Escape sequences in C are special character combinations that begin with a backslash (\) and represent characters that are either non-printable, have special meaning, or cannot be typed directly into a string or character constant.

Escape sequences are used to:

- Control formatting in output (like new lines or tabs)
- Represent special characters (like quotes or backslashes)
- Handle non-printable characters (like bell or backspace)

2. How can you format the output in C using escape sequences?

In C programming, escape sequences are used within output functions like printf() to control the formatting of the displayed text. They help insert special characters such as new lines, tabs, quotes, etc., which cannot be typed directly.

Escape sequences are used to:

- Improve readability of output
- Create structured layouts (like tables)
- Include special characters in strings
- Control cursor movement (like backspace or carriage return)

Commonly Used Escape Sequences for Formatting

\n printf("Hello\nWorld");
\ tprintf("Name\tAge");
\\ printf("C:\\Users\\Daniya");
\" printf("\"C is cool\"");
\' printf("\'A\'");
\b printf("Helloo\b");
\r printf("12345\rAB");

3. What are some common escape sequences used in C programming?

Common Escape Sequences in C

\n	printf("Hello\nWorld");
\t	printf("Name\tAge");
\\	printf("C:\\Users\\Daniya");
\"	printf("\"C is cool\"");
\'	printf("\'A\'");
\b	printf("Helloo\b");
\r	printf("12345\rAB");
\a	printf("\a");
\0	char str[] = "Hi\0Bye";

Post-Lab Questions:

1. What escape sequences did you use in the program to format the output?

Escape Sequences Used:

\n Moves to a new line

2. How would you print a backslash (\) in your program using escape sequences?

To print a backslash, you use double backslashes: \\

eg; printf("Path: C:\\Users\\Daniya\\Documents\n");

3. How would you modify the program to include more fields, such as phone number, using proper formatting?

You can simply add more printf() lines using the same formatting style with \t and \n.

```
#include <stdio.h>
int main()
{ printf("Student\t\t:\tDaniya\n");
 printf("Course\t\t:\tB.Tech\n");
 printf("Phone Number\t:\t+91-9876543210\n");
 printf("Quote\t\t:\t\"Keep learning!\"\n"); return 0; }
```

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:

main.c

```c
#include <stdio.h>
int main() {
    float physics, chemistry, math, average;
    // Input PCM marks
    printf("Enter Physics marks: ");
    scanf("%f", &physics);
    printf("Enter Chemistry marks: ");
    scanf("%f", &chemistry);
    printf("Enter Math marks: ");
    scanf("%f", &math);
    // Calculate average
    average = (physics + chemistry + math) / 3;
    // Display average
    printf("\nAverage Marks: %.2f\n", average);
    // Use conditional operator to check eligibility
    (average >= 50) ? printf("Eligible for admission\n") : printf
        ("Not eligible for admission\n");
    return 0;
}
```

Output

```
Enter Physics marks: 57
Enter Chemistry marks: 89
Enter Math marks: 63

Average Marks: 69.67
Eligible for admission


=== Code Execution Successful ===
```

Pre-Lab Questions:

1. What data types will you use to store the marks in this program?

To store marks, we typically use:

✅ float or double

- Because marks can be decimal values (e.g., 87.5).
- float is usually enough, but double gives more precision.

Example:

float mark1, mark2, mark3;

If you're sure marks will always be whole numbers, you can use:

- int → for integer marks like 85, 90, etc.

2. What formula will you use to calculate the average of three numbers?

To calculate the average of three marks:

✅ Formula:

Average= {{mark1}+{mark2}+{mark3}}/3

Example in C:

float average = (mark1 + mark2 + mark3) / 3.0;

🔸 Use 3.0 instead of 3 to ensure floating-point division.

3. How will you implement conditional logic to check eligibility in this program?

You can use an if-else statement to check if the average meets the eligibility criteria.

✅ Example:

if (average >= 40.0) { printf("Eligible\n"); } else { printf("Not Eligible\n"); }

You can also add more conditions:

if (average >= 75.0) { printf("Distinction\n"); } else if (average >= 60.0) { printf("First Class\n"); } else if (average >= 40.0) { printf("Pass\n"); } else { printf("Fail\n"); }

Post-Lab Questions:

1. What was the output of the program when you tested it with different marks?

```
#include <stdio.h>
int main() {
 float m1 = 75, m2 = 80, m3 = 85;
    float avg = (m1 + m2 + m3) / 3.0;
printf("Average = %.2f\n", avg);
if (avg >= 40)
 printf("Eligible\n");
    else
  printf("Not Eligible\n");
 return 0 }
```

by using different values we get different avg. hence some are eligible and some are not

2. What would happen if you input negative marks? How can you handle such cases in the code?

If you enter negative marks (e.g., -10), the program will still calculate the average and might give incorrect or misleading results.

✅ Solution: Input Validation

You can add a check to ensure marks are non-negative and within a valid range (e.g., 0 to 100).

3. How can you modify the program to get input for more subjects or add more conditions for eligibility?

To Add More Subjects:

- Add more variables (e.g., m4, m5)
- Update the average formula

```
float m1, m2, m3, m4, m5;
 printf("Enter marks for 5 subjects: ");
 scanf("%f %f %f %f %f", &m1, &m2, &m3, &m4, &m5);
float avg = (m1 + m2 + m3 + m4 + m5) / 5.0;
```

To Add More Eligibility Conditions:

Use if-else if blocks for categories like:

```
if (avg >= 75) printf("Distinction\n");
else if (avg >= 60) printf("First Class\n");
 else if (avg >= 50) printf("Second Class\n");
 else if (avg >= 40) printf("Pass\n");
 else printf("Fail\n");
```

Screenshots: Screenshot to show the code is uploaded on Github

Conclusion:

In C programming, escape sequences play a vital role in formatting output, making it clean, readable, and professional. By using sequences like \n, \t, \\, and \", we can control how text appears on the screen and include special characters that aren't directly typable.
When building a program to handle student marks, we use appropriate data types like float to store decimal values, apply the correct average formula, and implement conditional logic to determine eligibility. Adding input validation ensures the program handles errors like negative marks gracefully, and expanding the logic allows for more subjects and detailed classifications like "Distinction" or "First Class."
Altogether, these concepts help you write robust, user-friendly programs that reflect real-world academic scenarios—an essential skill for any aspiring engineer or developer.

CO's Covered:

lab covered how to print different things using escape sequence

……
……

……
….
……
…..

Expt. No. 2

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

#	Rubrics	Points
1	Code Functionality (0 – 3)	
2	Code Readability & Structure (0 – 3)	
3	Plagiarism (0 – 3)	
4	Outputs (0 – 3)	
5	Source Code Versioning (0 – 1)	
	Total Points	

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?

To sum even numbers instead of odd numbers, you have two simple options in a C program:

Option 1: Change the if condition (Easier to implement)
Change the modulo check inside your loop to look for a remainder of zero:

```
for (i = start; i <= end; i++) {
if (i % 2 == 0) { // Check if remainder is 0 (even)
sum += i;
}
}
```

Option 2: Change loop iteration (More efficient)
Start the loop at the first even number in the range and increment by 2:

```
// Ensure start is the minimum valueif (start % 2 != 0) {
start++;
}
for (i = start; i <= end; i += 2) {
sum += i;
}
```

How can you modify the condition inside the loop to sum even numbers?
To modify the condition inside a loop to sum even numbers in a C program , you need to check if the current iteration variable i in this example) is divisible by 2 with no remainder.
You can achieve this using the modulo operator (%).
To (summing evens):

```
if (i % 2 == 0) {
sum += i;
}
```

ANS 2 To handle cases where the starting number is larger than the ending number and ensure the program works regardless of the input order, you should first normalize the inputs so that the start variable always holds the minimum value and the end variable always holds the maximum value.
You can achieve this in two common ways:
Option 1: Swap the variables (Recommended)
Before the loop begins, add an if statement to check if the inputs are in the wrong order. If they are, use a temporary variable to swap their values.
This ensures the loop condition (i <= end) always works correctly because start will never be greater than end.

```
int start_num, end_num, temp;

// ... get inputs from user into start_num and end_num ...// Swap the numbers if they are in the wrong orderif (start_num
> end_num) {
temp = start_num;
start_num = end_num;
end_num = temp;
}
// Now the loop can safely run from start_num to end_numfor (i = start_num; i <= end_num; i++) {
// ... loop logic to sum numbers ...
}
```

Option 2: Use min() and max() functions
If your programming environment supports standard library functions like min() and max() (often found in <stdlib.h> or <algorithm> in C++), you can determine the correct loop boundaries regardless of input order.

int start_num, end_num;
// ... get inputs from user ...int minVal = min(start_num, end_num);
int maxVal = max(start_num, end_num);
// Use the determined min and max values in the loopfor (i = minVal; i <= maxVal; i++) {
// ... loop logic to sum numbers ...
}

ANS 3 When the starting and ending numbers are the same (e.g., the user enters 5 as the start and 5 as the end):

Output

The output of the program will be that single number itself. If the input is 5 and 5, the program will output 5 (assuming the program is summing all numbers in the range).

Program Behavior

Initialization: The loop counter i is initialized to the starting number (e.g., i = 5).

Condition Check: The loop condition (i <= end) is checked. In this case, 5 <= 5 is true, so the loop body executes.

Execution: The current value of i (which is 5) is added to the running total sum.

Increment/Update: The loop counter i is incremented (e.g., i becomes 6).

Termination: The loop condition is checked again: 6 <= 5 is false. The loop terminates.

Result: The program finishes and prints the sum, which contains only the single input number.

Is this behavior expected?

Yes, this behavior is expected.

The range between a number and itself, inclusively, contains only that single number. The loop condition correctly interprets a range of [5, 5] as containing only one iteration. The program functions as designed for a valid (albeit very short) range of numbers.

ANS4 To avoid counting or summing negative numbers, you can add a simple check within the loop using an if statement. The continue statement can be used to skip the rest of the current loop iteration immediately if the number is negative.

for (i = start; i <= end; i++) {
if (i < 0) {
continue; // Skip the rest of the loop body for negative numbers
}
// Only non-negative numbers reach this point
sum += i;
}

Alternatively, simply wrap your existing summation logic inside an if block that only runs if the number is non-negative:

for (i = start; i <= end; i++) {
if (i >= 0) {
sum += i;
}
}

How to ensure only positive odd numbers are considered

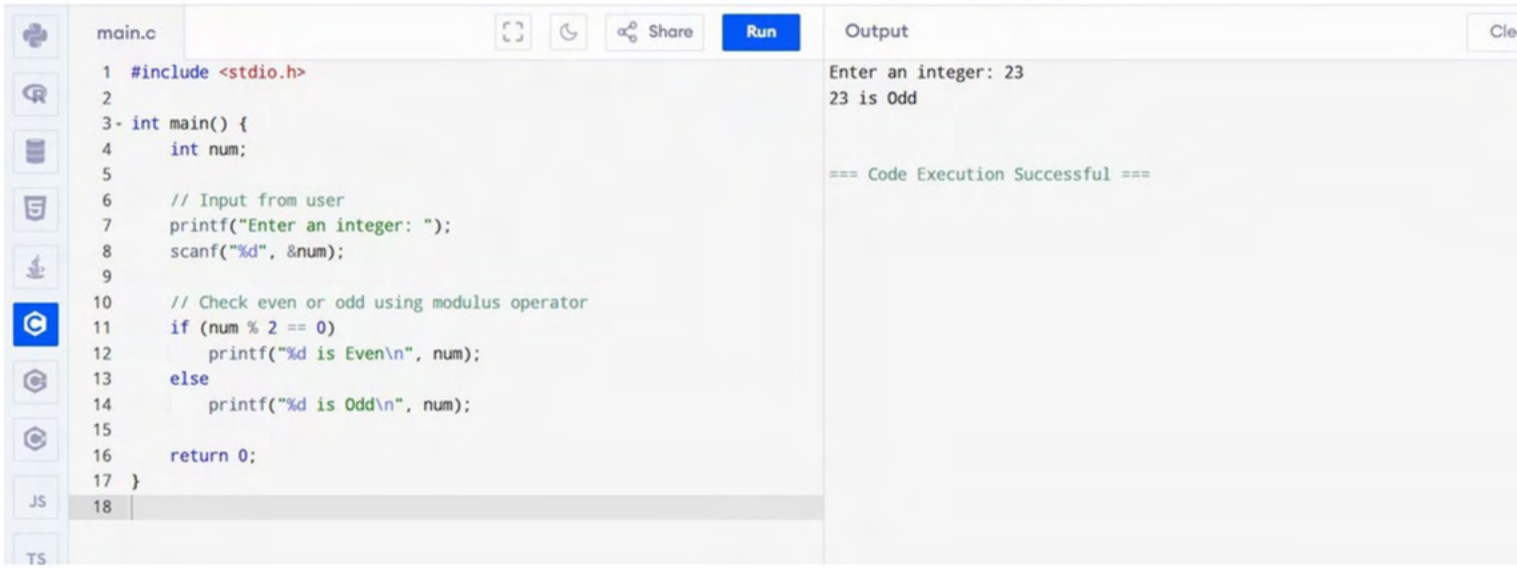
To restrict the sum to only positive odd numbers, you need a combined condition that checks for two criteria:

The number must be greater than zero (i > 0).

The number must be odd (i % 2 != 0).

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:

main.c

```c
#include <stdio.h>
int main() {
    int start, end, sum = 0;
    // Input range from user
    printf("Enter the starting number: ");
    scanf("%d", &start);
    printf("Enter the ending number: ");
    scanf("%d", &end);
    // Loop through the range
    for (int i = start + 1; i < end; i++) {
        if (i % 2 != 0) {
            sum += i;
        }
    }
    // Output the result
    printf("Sum of odd numbers between %d and %d is: %d\n", start,
        end, sum);
    return 0;
}
```

Output

```
Enter the starting number: 34
Enter the ending number: 56
Sum of odd numbers between 34 and 56 is: 495


=== Code Execution Successful ===
```

Screenshots: Screenshot to show the code is uploaded on Github

Conclusion:

……

……

CO's Covered:

……

……

Expt. No. 3

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

#	Rubrics	Points
1	Code Functionality (0 – 3)	
2	Code Readability & Structure (0 – 3)	
3	Plagiarism (0 – 3)	
4	Outputs (0 – 3)	
5	Source Code Versioning (0 – 1)	
	Total Points	

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
```

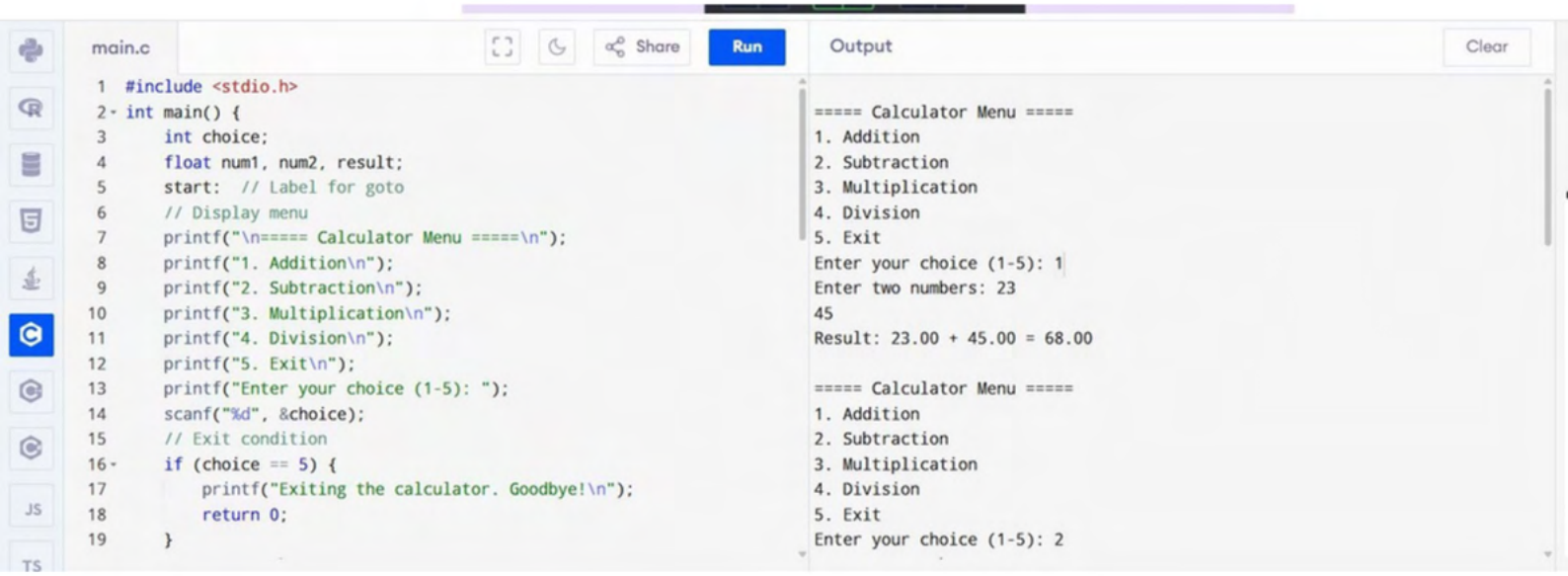
- **goto**: This keyword is followed by the name of a label, and execution jumps to that label.
- **Label**: A user-defined identifier followed by a colon (:) that marks the destination of the goto statement.

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:



main.c Share Run Output Clear

```c
    // Input numbers
    printf("Enter two numbers: ");
    scanf("%f %f", &num1, &num2);
// Perform operation based on choice
    switch (choice) {
        case 1:
            result = num1 + num2;
            printf("Result: %.2f + %.2f = %.2f\n", num1, num2,
                result);
            break;
        case 2:
            result = num1 - num2;
            printf("Result: %.2f - %.2f = %.2f\n", num1, num2,
                result);
            break;
        case 3:
            result = num1 * num2;
            printf("Result: %.2f * %.2f = %.2f\n", num1, num2,
                result);
```

```
Enter two numbers: 27
13
Result: 27.00 - 13.00 = 14.00

===== Calculator Menu =====
1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Exit
Enter your choice (1-5): 3
Enter two numbers: 23
12
Result: 23.00 * 12.00 = 276.00

===== Calculator Menu =====
1. Addition
2. Subtraction
3. Multiplication
```

main.c Share Run Output Clear

```c
            break;
        case 4:
            if (num2 != 0)
                result = num1 / num2;
            else {
                printf("Error: Division by zero is not allowed.\n");
                goto start;
            }
            printf("Result: %.2f / %.2f = %.2f\n", num1, num2,
                result);
            break;
        default:
            printf("Invalid choice. Please try again.\n");
    }
    // Go back to menu
    goto start;
    return 0;
}
```

```
3. Multiplication
4. Division
5. Exit
Enter your choice (1-5): 4
Enter two numbers: 25
5
Result: 25.00 / 5.00 = 5.00

===== Calculator Menu =====
1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Exit
Enter your choice (1-5): 5
Exiting the calculator. Goodbye!


=== Code Execution Successful ===
```

Screenshots: Screenshot to show the code is uploaded on Github

Conclusion:

……

……

CO's Covered:

……

……

Expt. No. 4

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

#	Rubrics	Points
1	Code Functionality (0 – 3)	
2	Code Readability & Structure (0 – 3)	
3	Plagiarism (0 – 3)	
4	Outputs (0 – 3)	
5	Source Code Versioning (0 – 1)	
	Total Points	

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 functiondeclarationspecifiesthefunction'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. **StandardLibrary 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:

- **ReturnType** : 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** :Functionsbreakthe program into smaller, manageable sections.
- **Reusability**:Oncedefined,afunction 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)**

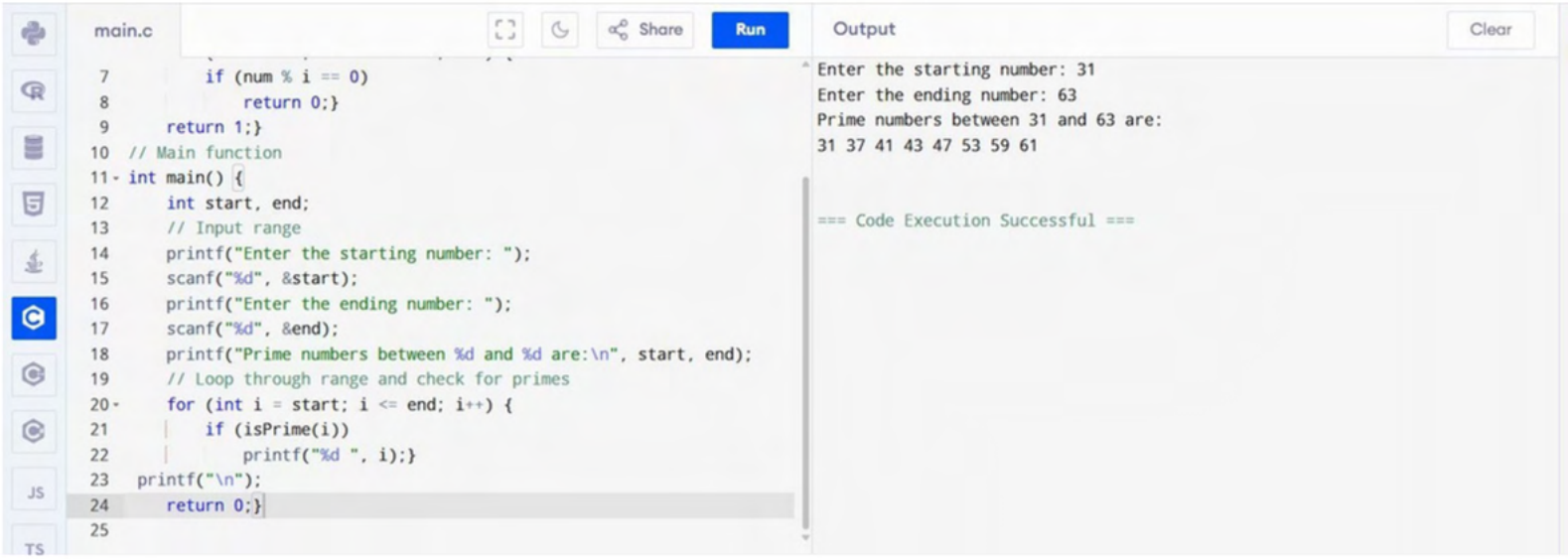
Program with Output:

main.c

```c
#include <stdio.h>
// Function to check if a number is prime
int isPrime(int num) {
    if (num <= 1)
        return 0;
    for (int i = 2; i <= num / 2; i++) {
        if (num % i == 0)
            return 0;}
    return 1;}
// Main function
int main() {
    int start, end;
    // Input range
    printf("Enter the starting number: ");
    scanf("%d", &start);
    printf("Enter the ending number: ");
    scanf("%d", &end);
    printf("Prime numbers between %d and %d are:\n", start, end);
    // Loop through range and check for primes
```

Output

```
Enter the starting number: 31
Enter the ending number: 63
Prime numbers between 31 and 63 are:
31 37 41 43 47 53 59 61


=== Code Execution Successful ===
```


main.c
Share
Run
Output
Clear
if (num % i == 0)
return 0;}
return 1;}
// Main function
int main() {
int start, end;
// Input range
printf("Enter the starting number: ");
scanf("%d", &start);
printf("Enter the ending number: ");
scanf("%d", &end);
printf("Prime numbers between %d and %d are:\n", start, end);
// Loop through range and check for primes
for (int i = start; i <= end; i++) {
if (isPrime(i))
printf("%d ", i);}
printf("\n");
return 0;}
Enter the starting number: 31
Enter the ending number: 63
Prime numbers between 31 and 63 are:
31 37 41 43 47 53 59 61
=== Code Execution Successful ===

Screenshots: Screenshot to show the code is uploaded on Github

Conclusion:

……

……

CO's Covered:

……

……

Expt. No. 5

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

#	Rubrics	Points
1	Code Functionality (0 – 3)	
2	Code Readability & Structure (0 – 3)	
3	Plagiarism (0 – 3)	
4	Outputs (0 – 3)	
5	Source Code Versioning (0 – 1)	
	Total Points	

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 codeuntilaspecificcondition 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:

main.c

```c
#include <stdio.h>

// Recursive function to find factorial
long long factorial_recursive(int n) {
    if (n == 0 || n == 1)
        return 1;
    else
        return n * factorial_recursive(n - 1);
}

// Iterative function to find factorial
long long factorial_iterative(int n) {
    long long fact = 1;
    for (int i = 1; i <= n; i++) {
        fact *= i;
    }
    return fact;
}

int main() {
    int num;
```

Output

```
Enter a positive integer: 5
Factorial (Iterative) of 5 = 120
Factorial (Recursive) of 5 = 120


=== Code Execution Successful ===
```

main.c

```c
    }
    return fact;
}

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 (Iterative) of %d = %lld\n", num,
            factorial_iterative(num));
        printf("Factorial (Recursive) of %d = %lld\n", num,
            factorial_recursive(num));
    }

    return 0;
}
```

Output

```
Enter a positive integer: 5
Factorial (Iterative) of 5 = 120
Factorial (Recursive) of 5 = 120


=== Code Execution Successful ===
```

Screenshots: Screenshot to show the code is uploaded on Github

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.*

#	Rubrics	Points
1	Code Functionality (0 – 3)	
2	Code Readability & Structure (0 – 3)	
3	Plagiarism (0 – 3)	
4	Outputs (0 – 3)	
5	Source Code Versioning (0 – 1)	
	Total Points	

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** forlocalvariables declared inside functions or blocks.
- **Scope**:Localtothefunction/block where it is declared.
- **Lifetime**:Existsonlyduringtheexecution of the block/function. It is destroyed when theblock/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)

- Usedtostore variables in the **CPU register** for faster access, instead of RAM. Recommended for variables that are frequentlyused.
- **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 tothefunction/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

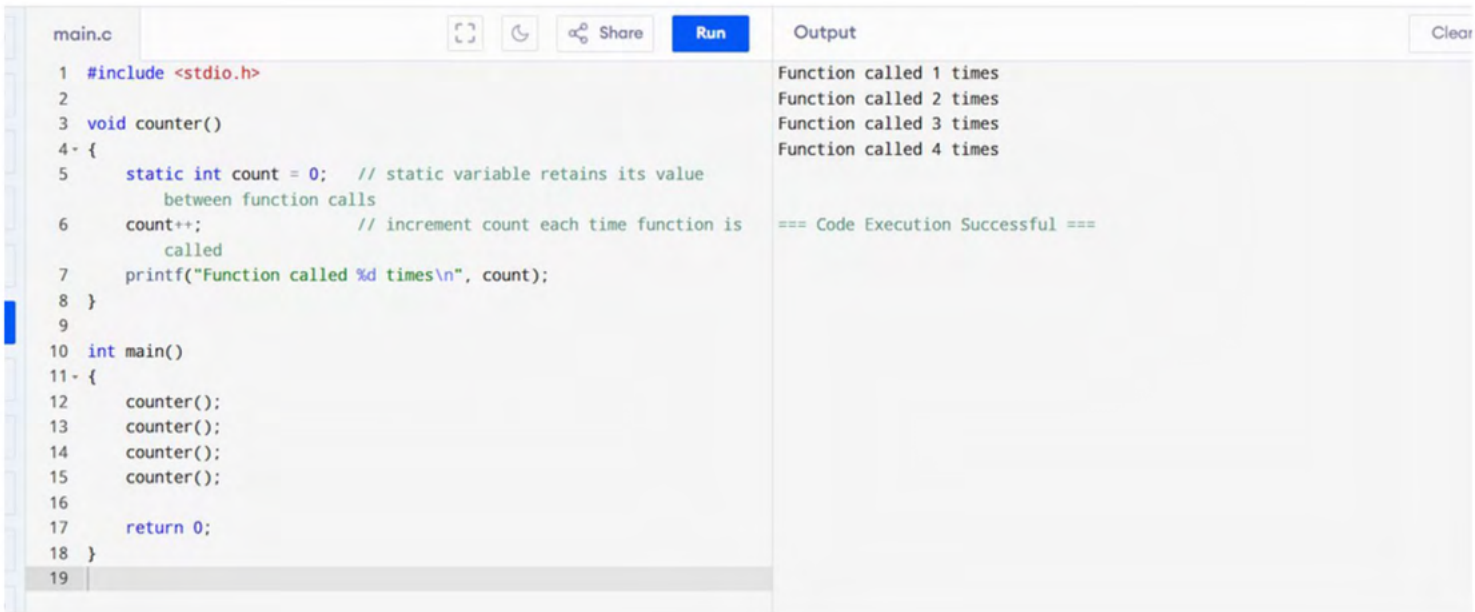
int x = 10; // Definition

// 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:



Screenshots: Screenshot to show the code is uploaded on Github

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.*

#	Rubrics	Points
1	Code Functionality (0 – 3)	
2	Code Readability & Structure (0 – 3)	
3	Plagiarism (0 – 3)	
4	Outputs (0 – 3)	
5	Source Code Versioning (0 – 1)	
	Total Points	

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 sizedetermineshowmany 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**:

   o **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**:Arraysstore data in contiguous memory locations, allowing fast access to elementsusingindices.
- **Organized storage**: You can manage multiple variables of the same type under one name.
- **Easy iteration**:Arrayscan be easily traversed using loops.

Disadvantages of Arrays:

- **Fixed size**:Oncedeclared, 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:

main.c

Share Run

```c
#include <stdio.h>

int main()
{
    int arr[50], n, i, max;

    printf("Enter number of elements: ");
    scanf("%d", &n);

    printf("Enter %d elements:\n", n);
    for(i = 0; i < n; i++)
    {
        scanf("%d", &arr[i]);
    }

    max = arr[0];  // Assume first element is the largest

    for(i = 1; i < n; i++)
    {
        if(arr[i] > max)
        {
```

Output Clear

```
Enter number of elements: 5
Enter 5 elements:
4
5
7
8

3
Largest element in the array is: 8


=== Code Execution Successful ===
```

main.c

Share Run

```c
    printf("Enter %d elements:\n", n);
    for(i = 0; i < n; i++)
    {
        scanf("%d", &arr[i]);
    }

    max = arr[0];  // Assume first element is the largest

    for(i = 1; i < n; i++)
    {
        if(arr[i] > max)
        {
            max = arr[i];   // Update max if current element is
                larger
        }
    }

    printf("Largest element in the array is: %d\n", max);

    return 0;
}
```

Output Clear

```
Enter number of elements: 5
Enter 5 elements:
4
5
7
8

3
Largest element in the array is: 8


=== Code Execution Successful ===
```

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

Program with Output:

main.c

```c
#include <stdio.h>
int main() {
    int rows, cols;
    // Input matrix dimensions
    printf("Enter number of rows and columns: ");
    scanf("%d %d", &rows, &cols);
    int matrix1[rows][cols], matrix2[rows][cols], sum[rows][cols];
    // Input elements of first matrix
    printf("Enter elements of Matrix 1:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            scanf("%d", &matrix1[i][j]);
        }
    }
    // Input elements of second matrix
    printf("Enter elements of Matrix 2:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            scanf("%d", &matrix2[i][j]);
        }
    }
```

Output

```
Enter number of rows and columns: 2
3
Enter elements of Matrix 1:
2
5

7
7
8
9
Enter elements of Matrix 2:

4
6
8
9
4
2
Sum of the two matrices:
6   11  15
16  12  11
```

main.c

```c
        for (int j = 0; j < cols; j++) {
            scanf("%d", &matrix2[i][j]);
        }
    }
    // Calculate sum of matrices
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            sum[i][j] = matrix1[i][j] + matrix2[i][j];
        }
    }
    // Display the result
    printf("Sum of the two matrices:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%d\t", sum[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Output

```
2
5

7
7
8
9
Enter elements of Matrix 2:

4
6
8
9
4
2
Sum of the two matrices:
6   11  15
16  12  11


=== Code Execution Successful ===
```

Screenshots: Screenshot to show the code is uploaded on Github

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.*

#	Rubrics	Points
1	Code Functionality (0 – 3)	
2	Code Readability & Structure (0 – 3)	
3	Plagiarism (0 – 3)	
4	Outputs (0 – 3)	
5	Source Code Versioning (0 – 1)	
	Total Points	

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 mustbesufficienttohold 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.

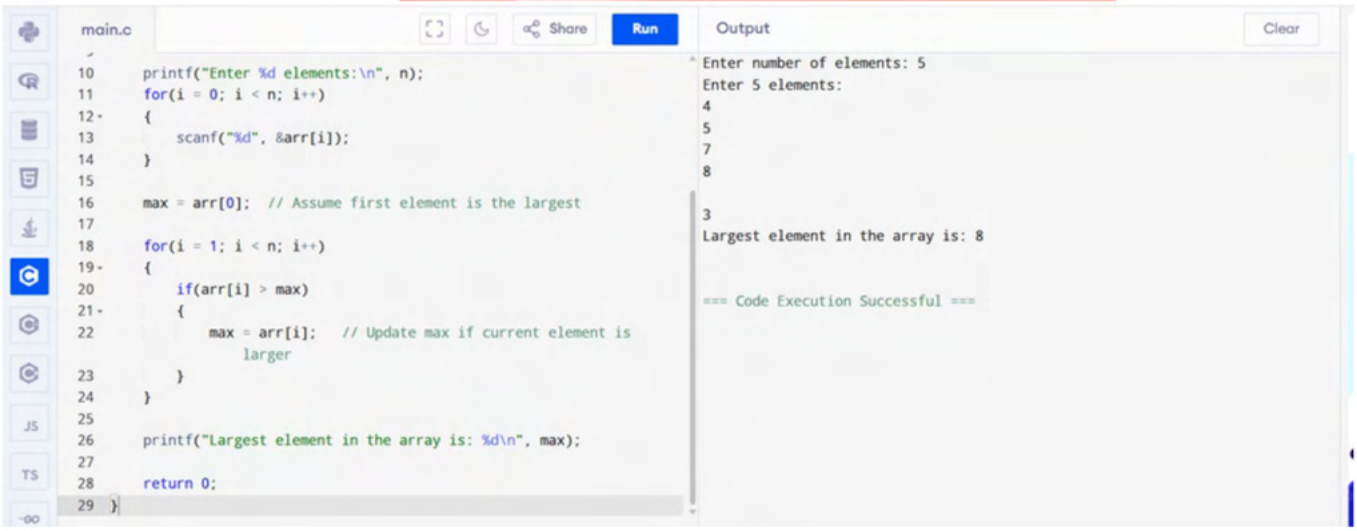
  int result = strcmp(str1, str2);

String Handling Challenges in C:

- **Fixed size**:Oncethesize of thearray is declared, it cannot be changed dynamically. Thiscanlimitflexibility.
- **Manual management** :C does not provide native string types like modern languages. Youmustmanage 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:

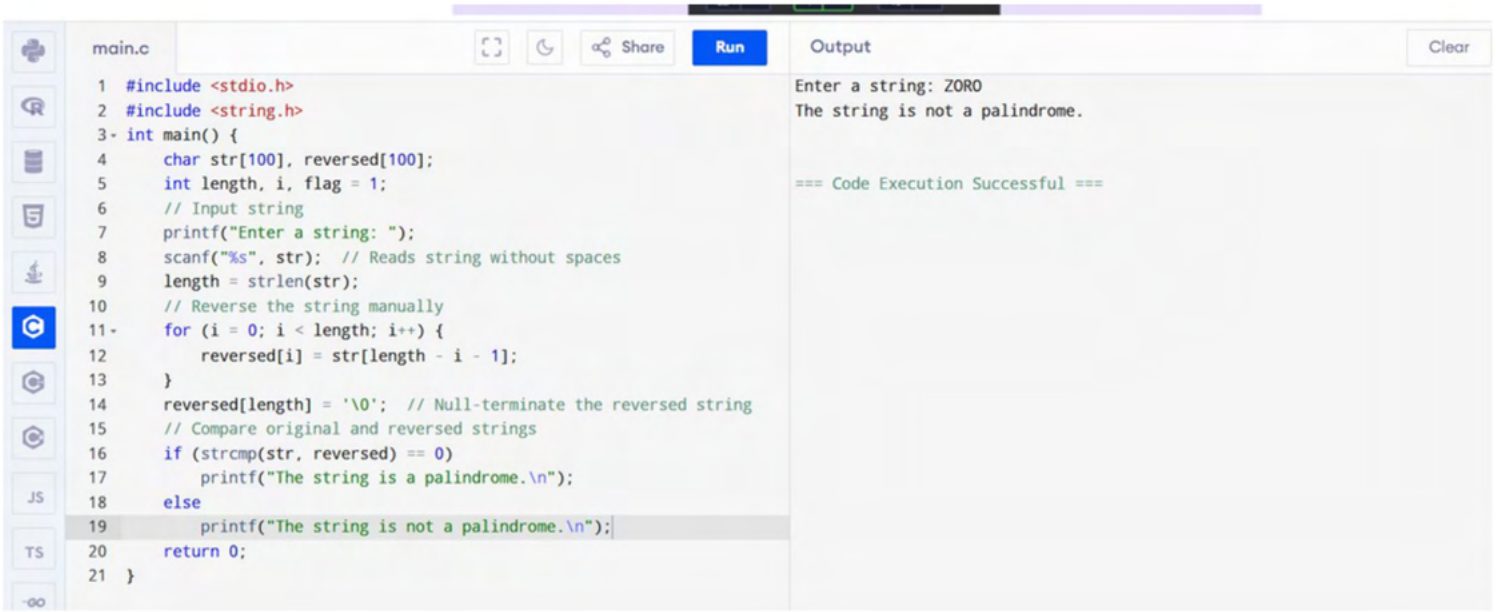


```c
    printf("Enter %d elements:\n", n);
    for(i = 0; i < n; i++)
    {
        scanf("%d", &arr[i]);
    }

    max = arr[0];  // Assume first element is the largest

    for(i = 1; i < n; i++)
    {
        if(arr[i] > max)
        {
            max = arr[i];   // Update max if current element is
                larger
        }
    }

    printf("Largest element in the array is: %d\n", max);

    return 0;
}
```

```
Enter number of elements: 5
Enter 5 elements:
4
5
7
8

3
Largest element in the array is: 8


=== Code Execution Successful ===
```

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

Program with Output:



```c
#include <stdio.h>
#include <string.h>
int main() {
    char str[100], reversed[100];
    int length, i, flag = 1;
    // Input string
    printf("Enter a string: ");
    scanf("%s", str);  // Reads string without spaces
    length = strlen(str);
    // Reverse the string manually
    for (i = 0; i < length; i++) {
        reversed[i] = str[length - i - 1];
    }
    reversed[length] = '\0';  // Null-terminate the reversed string
    // Compare original and reversed strings
    if (strcmp(str, reversed) == 0)
        printf("The string is a palindrome.\n");
    else
        printf("The string is not a palindrome.\n");
    return 0;
}
```

```
Enter a string: ZORO
The string is not a palindrome.


=== Code Execution Successful ===
```

Screenshots: Screenshot to show the code is uploaded on Github

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.*

#	Rubrics	Points
1	Code Functionality (0 – 3)	
2	Code Readability & Structure (0 – 3)	
3	Plagiarism (0 – 3)	
4	Outputs (0 – 3)	
5	Source Code Versioning (0 – 1)	
	Total Points	

Performed On: _____

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 (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 thestructurenameand 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:

main.c

```c
#include <stdio.h>
#include <string.h>
#define SIZE 5
// Define the structure
struct student_record {
    char name[50];
    int roll_number;
    float total_marks;
};
int main() {
    struct student_record students[SIZE];
    int i, topper_index = 0;
    // Input data for 5 students
    for (i = 0; i < SIZE; i++) {
        printf("\nEnter details for Student %d:\n", i + 1);
        printf("Name: ");
        scanf("%s", students[i].name);
        printf("Roll Number: ");
        scanf("%d", &students[i].roll_number);
        printf("Total Marks: ");
        scanf("%f", &students[i].total_marks);
    }
    // Find the topper
    for (i = 1; i < SIZE; i++) {
        if (students[i].total_marks > students[topper_index]
            .total_marks) {
            topper_index = i;
        }
    }
    // Display topper details
    printf("\n Topper Details:\n");
    printf("Name        : %s\n", students[topper_index].name);
    printf("Roll Number : %d\n", students[topper_index].roll_number
        );
    printf("Total Marks : %.2f\n", students[topper_index]
        .total_marks);
    return 0;
}
```

Output

```
Enter details for Student 1:
Name: LUFFY
Roll Number: 21
Total Marks: 99

Enter details for Student 2:
Name: NAMI
Roll Number: 31
Total Marks: 67

Enter details for Student 3:
Name: ZORO
Roll Number: 45
Total Marks: 83

Enter details for Student 4:
Name: SANJI
Roll Number: 49
Total Marks: 80

Enter details for Student 5:
Name: ROBIN
Roll Number: 36
Total Marks: 90

 Topper Details:
Name        : LUFFY
Roll Number : 21
Total Marks : 99.00


=== Code Execution Successful ===
```

Screenshots: Screenshot to show the code is uploaded on Github

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.*

#	Rubrics	Points
1	Code Functionality (0 – 3)	
2	Code Readability & Structure (0 – 3)	
3	Plagiarism (0 – 3)	
4	Outputs (0 – 3)	
5	Source Code Versioning (0 – 1)	
	Total Points	

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 typeofthepointermust 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 performanceforcertainoperations.
- **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:

main.c

```c
#include <stdio.h>
int main() {
    int a, b, sum;
    int *p1, *p2;   // pointer variables

    // Input two numbers
    printf("Enter first number: ");
    scanf("%d", &a);

    printf("Enter second number: ");
    scanf("%d", &b);
// Assign addresses of variables to pointers
    p1 = &a;
    p2 = &b;
    // Add the values pointed to by p1 and p2
    sum = *p1 + *p2;
    printf("Sum = %d\n", sum);
    return 0;
}
```

Output

```
Enter first number: 23
Enter second number: 43
Sum = 66

=== Code Execution Successful ===
```

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

Program with Output:

main.c

Share Run

```c
#include <stdio.h>
int main() {
    int arr[100], n;
    int *ptr;
    // Input array size
    printf("Enter number of elements: ");
    scanf("%d", &n);
    // Input array elements
    printf("Enter %d elements:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    // Set pointer to the last element
    ptr = &arr[n - 1];
    // Print elements in reverse using pointer
    printf("Array in reverse order:\n");
    for (int i = n - 1; i >= 0; i--) {
        printf("%d ", *ptr);
        ptr--;  // Move pointer backward
    }
    return 0;}
```

Output Clear

```
Enter number of elements: 4
Enter 4 elements:
6
7
4
2
Array in reverse order:
2 4 7 6

=== Code Execution Successful ===
```

Screenshots: Screenshot to show the code is uploaded on Github

Conclusion:

……

……

CO's Covered:

……

……

Expt. No. 11

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

#	Rubrics	Points
1	Code Functionality (0 – 3)	
2	Code Readability & Structure (0 – 3)	
3	Plagiarism (0 – 3)	
4	Outputs (0 – 3)	
5	Source Code Versioning (0 – 1)	
	Total Points	

Performed On: _____

Sign:_____

Experiment No. 11

CPL (C Programming Lab)

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

Software :

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 fileis created.
   - "a": Open a file for **appending** .Dataiswritten 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**.Itoverwrites 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:

main.c

```c
#include <stdio.h>
#include <stdlib.h>
struct Employee {
    int id;
    char name[50];
    float salary;
};
int main() {
    struct Employee emp;
    FILE *fp;
    int n;
    // Open file in write mode
    fp = fopen("employee_data.txt", "w");
    if (fp == NULL) {
        printf("Error opening file!\n");
        return 1;
    }
    // Input number of employees
    printf("Enter number of employees: ");
    scanf("%d", &n);
    // Write employee data to file
```

Output

```
Error opening file!


=== Code Exited With Errors ===
```

main.c

```c
    // Write employee data to file
    for (int i = 0; i < n; i++) {
        printf("\nEnter details for Employee %d:\n", i + 1);
        printf("ID: ");
        scanf("%d", &emp.id);
        printf("Name: ");
        scanf("%s", emp.name);
        printf("Salary: ");
        scanf("%f", &emp.salary);
        fwrite(&emp, sizeof(emp), 1, fp);
    }
    fclose(fp);
    // Reopen file in read mode
    fp = fopen("employee_data.txt", "r");
    if (fp == NULL) {
        printf("Error reading file!\n");
        return 1;
    }
    printf("\n Employee Records:\n");
    while (fread(&emp, sizeof(emp), 1, fp)) {
        printf("ID: %d\tName: %s\tSalary: %.2f\n", emp.id, emp.name,
```

Output

```
Error opening file!


=== Code Exited With Errors ===
```

main.c

```c
        scanf("%s", emp.name);
        printf("Salary: ");
        scanf("%f", &emp.salary);
        fwrite(&emp, sizeof(emp), 1, fp);
    }
    fclose(fp);
    // Reopen file in read mode
    fp = fopen("employee_data.txt", "r");
    if (fp == NULL) {
        printf("Error reading file!\n");
        return 1;
    }
    printf("\n Employee Records:\n");
    while (fread(&emp, sizeof(emp), 1, fp)) {
        printf("ID: %d\tName: %s\tSalary: %.2f\n", emp.id, emp.name,
            emp.salary);
    }
    fclose(fp);
    return 0;
}
```

Output

```
Error opening file!


=== Code Exited With Errors ===
```

Screenshots: Screenshot to show the code is uploaded on Github

Conclusion:

……

……

CO's Covered:

……

……

ASSIGNMENTS

Assignment. 1

Question:

.

#	Rubrics	Points
1	Understanding of Concepts (0 – 2)	
2	Correctness of Answer (0 – 2)	
3	Plagiarism (0 – 2)	
	Total Points	

Submitted On: _____

Sign:_____

Assignment. 2

Question:

.

#	Rubrics	Points
1	Understanding of Concepts (0 – 2)	
2	Correctness of Answer (0 – 2)	
3	Plagiarism (0 – 2)	
	Total Points	

Submitted On: _____

Sign:_____

Assignment. 3

Question:

.

#	Rubrics	Points
1	Understanding of Concepts (0 – 2)	
2	Correctness of Answer (0 – 2)	
3	Plagiarism (0 – 2)	
	Total Points	

Submitted On: _____

Sign:_____

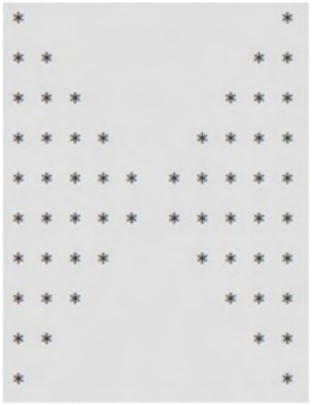
APPENDIX

APPENDIX 1

Additional List of Programs for Practice

Week 1

- Task 1 : WAP to print your name on screen.
- Task 2 : WAP to print the following pattern using printf.



- Task 3 : WAP to print the following table using printf.

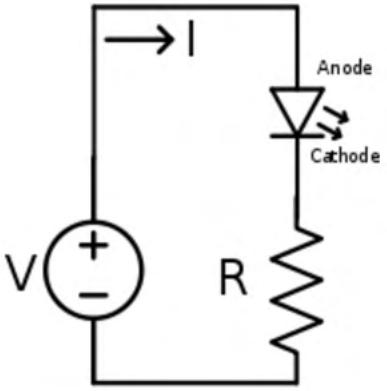
Roll No.	Name of Student	Marks
1	Jon Doe	35
2	Ranchordas Chanchad	37
3	Sanjay Singh	56

- Task 4 : WAP to print a letter.

```
To,
The Principal,
Rizvi College of Engineering,
Bandra (W).


        Subject : Requesting some books for the library,


Dear Sir/Madam,
        Myself john would like to request purchase the following books for the library.

_____________________________________________________________________________
|Title                                          |Author                 |Publication    |
|_______________________________________________|_______________________|_______________|
|C Programming                                  |Balguruswami           |McGraw Hill    |
|_______________________________________________|_______________________|_______________|
|Python Programming                             |Simon Says             |Pearsons       |
|_______________________________________________|_______________________|_______________|


Thanking you in anticipation.
Regards,
John Doe.
```

Week 2

- Task 5 : WAP to calculate the area of square.
- Task 6 : WAP to calculate the simple interest taking principal,rate of interest and no. of years as inputs from user.
- Task 7 : WAP to calculate the sum and average of three numbers.
- Task 8 : WAP to calculate perimeter and area of a circle with the help of constants.
- Task 9 : WAP to calculate the resistance required (R) to be connected in series with the LED to protect it from blowing up when Voltage (V) is applied.



Week 3

- Task 10 : WAP to calculate the gross salary of the employee given the basic salary. Travelling Allowance, Dearness Allowance and Home Rental Allowance is 20%, 80% and 15% of Basic Salary respectively.
- Task 11 : WAP to swap two numbers with the help of a Temporary Number.
- Task 12 : WAP to swap two numbers without the help of a Temporary Number.
- Task 13 : WAP to check if the number is even or not using conditional operator.
- Task 14 : WAP to find the greatest number amongst three entered numbers using conditional operator.
- Task 15 : WAP to find sum of two integers without using '+' operator.

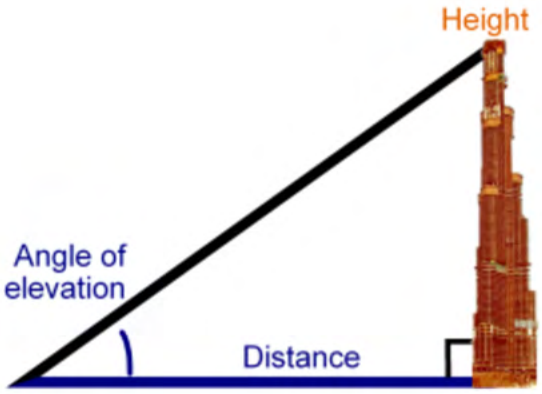
Week 4

- Task 16 : Write an algorithm and program to find if the entered character is a vovel or not.
- Task 17 : Draw a flowchart and program to convert a lower case character to uppercase and vise-versa.
- Task 18 : WAP to check if the year entered is a leap year or not.
- Task 19 : WAP to find the body mass index of the user when user enters height and weight. Also find the BMI category to which the user belongs with elseif ladder.
    - Underweight = <18.5
    - Normal weight = 18.5–24.9
    - Overweight = 25–29.9
    - Obesity = BMI of 30 or greater

- Task 20 : WAP to read three integer values from the keyboard and displays the output stating that they are sides of right angled triangle or not.

Week 5

- Task 21 : WAP to find the roots of the quadratic equation with real roots using math library.
- Task 22 : WAP to read weekday number and print weekday name using switch.
- Task 23 : WAP to design calculator with basic operations using switch.
- Task 24 : Modify the previous program to run repeatedly along with a exit condition using goto.
- Task 25 : WAP to find the height of the building when the user enters the distance from the building and the angle of elevation in degrees.



Week 6

- Task 26 : WAP to find if the entered number is a prime number or not.
- Task 27 : WAP to find the sum and average of the numbers entered by the user.
- Task 28 : WAP to print Fibonacci Series till number entered by the user is reached.
- Task 29 : WAP to print the following pattern. List of Patterns
- Task 30 : WAP to the following series where user enters the value for n. Also compute the value of the series.

Week 7

- Task 31 : WAP to find the LCM and GCD of two given numbers using while loop.
- Task 32 : Write a C program to find if the entered three digit number is an Armstrong number or not using do while loop.
- Task 33 : Write a program in C to convert a decimal number into binary using while loop.
- Task 34 : WAP to convert binary number to decimal number using while loop.
- Task 35 : Write a C program to find Strong Numbers within a range of numbers using do while loop.

Week 8

- Task 36 : Write a program in C to check a given number is even or odd using the functions.
- Task 37 : Write a program in C to print all the prime numbers between two numbers entered by the user using functions.
- Task 38 : WAP to find area of a circle and a hexagon and also find which shape has a larger area using functions.
- Task 39 : Write a program in C to find the Factorial of a number using recursion
- Task 40 : Write a program in C to Print Fibonacci Series using recursion
- Task 41 : Write a program to print pascal triangle using recursion function.

Week 9

- Task 42 : Write a program in C to read n number of values in an array and display it in reverse order Task 43 : Write a program in C to sort elements of the array in aescending order. Task 44
- : Write a program in C to find the second largest element in an array. Task 45 : Write a program
- in C for multiplication of two square Matrices. Task 46 : Write a program in C to calculate
- determinant of a 3 x 3 matrix. Task 47 : Write a program in C to find the length of a string
- without using library function Task 48 : Write a program in C to check if the entered string is
- palindrome or not. Task 49 : Write a program in C reverse a string using string.h library
-
-

Week 10

- Task 50 : Define a structure data type called time_struct containing three members integer hour, integer minute and integer second. Develop a program that would assign values to the individual members and start running the clock and disply time in HH:MM:SS format. Task 51 : Design a
- structure student_record to contain name, roll_number, and total marks obtained. Write a program to read 10 students data from the user and then display the first three toppers on the screen.

Week 11

- Task 52 : Write a program in C to add two numbers using pointers
- Task 53 : WAP to find the permutation and combination for a given value of n and r with the help of pointers.
- Task 54 : Write a program in C to print the elements of an array in reverse order using pointers.
- Task 55 : Using pointers, write a function that recieves a charachter string and a charachter as argument and deletes all occurences of this charachter in the string and prints the corrected string without any holes.

PPT for Output : https://rizvicoe.github.io/cptasks
Video Explanation : https://bit.ly/cprog19

APPENDIX 2

Important Resources for Coding Practice

- **https://www.hackerrank.com/** : HackerRank is a popular online platform that provides coding challenges and competitions for developers of all skill levels. It allows users to practice coding, improve problem-solving skills, and prepare for technical interviews by solving challenges in a variety of domains.
- **https://leetcode.com/** : LeetCode is a popular online platform that helps developers practice coding and prepare for technical interviews. It offers a vast collection of coding problems categorized by difficulty and topic, including data structures, algorithms, dynamic programming, and more.
- **https://www.codechef.com/practice/c** : CodeChef is an online competitive programming platform that offers a wide range of coding challenges and contests for developers of all levels. It features monthly contests like Long Challenge, Cook-Off, and LunchTime, where users can compete, improve problem-solving skills, and earn rankings.

Important Resources to Learn C Programming

- **https://learn-c.org/** : To learn C programming, you can visit Learn-C.org which offers a comprehensive and interactive tutorial to help you master C. It covers topics from basic syntax to advanced features, with exercises for hands-on practice. The site is ideal for beginners and intermediate learners, providing real-time coding environments so you can test your solutions immediately.

APPENDIX 3

Here are the download links for free C compilers:

1. **GCC (GNU Compiler Collection) –** https://gcc.gnu.org/install/
2. **Clang –** https://clang.llvm.org/get_started.html
3. **MinGW –** https://sourceforge.net/projects/mingw/
4. **Turbo C++ –** https://sourceforge.net/projects/turboc/
5. **TCC (Tiny C Compiler) –** https://bellard.org/tcc/
6. **Dev-C++ –** https://sourceforge.net/projects/orwelldevcpp/

Here's a list of popular IDEs for C programming:

1. **Visual Studio Code** – Lightweight and supports C with extensions.
   - https://code.visualstudio.com/
2. **Code::Blocks** – Free, open-source, customizable IDE.
   - https://www.codeblocks.org/
3. **Dev-C++** – Classic IDE for C/C++, based on GCC.
   - https://sourceforge.net/projects/orwelldevcpp/
4. **Eclipse IDE for C/C++ Developers** – Feature-rich IDE with powerful debugging.
   - https://www.eclipse.org/downloads/packages/release
5. **CLion** – Powerful IDE from JetBrains, includes advanced tools.
   - https://www.jetbrains.com/clion/

APPENDIX 4

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 Files**: 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**