DRONACHARYA
College of Engineering

LABORATORY MANUAL

B.Tech. Semester- I

PROGRAMMING FOR PROBLEM SOLVING USING C
Subject code: CSE-101P

Prepared by:	**Checked by:**	**Approved by:**
Prof. Sakshi Ahuja	Prof. Megha Goel	Name : Prof. (Dr.) Isha Malhotra
Sign.: ……	**Sign.: ……**	**Sign.: ……**

DEPARTMENT OF APPLIED SCIENCE & HUMANITIES
DRONACHARYA COLLEGE OF ENGINEERING
KHENTAWAS, FARRUKH NAGAR, GURUGRAM (HARYANA)

Table of Contents

Vision and Mission of the Institute

Vision:

"Empowering human values and advanced technical education to navigate and address global challenges with excellence".

Mission:

- **M1 -** Seamlessly integrate human values with advanced technical education.
- **M2 -** Supporting the cultivation of a new generation of innovators who are not only skilled but also ethically responsible.
- **M3 -** Inspire global citizens who are equipped to create positive and sustainable impact, driving progress towards a more inclusive and harmonious world.

Vision and Mission of the Department

Vision:

- To establish a strong foundation for first-year engineering students, aiming to equip them with the skills to innovate and devise engineering solutions.

Mission

- **M1**: To develop a solid foundation of knowledge and hands on experience in budding technocrats, empowering them to apply scientific principles to address complex engineering challenges.
- **M2**: To provide education that fosters comprehension and collaboration between engineering and other core field of Applied Sciences.
- **M3**: To inculcate values and ethics in students and make them responsible citizens of India.

Program Educational Objectives (PEOs)

- **PEO1:** PEO1: To instill the basic principles of Applied Sciences to enable students learn technical subjects effectively.
- **PEO2:** To equip students with innovative skills that improve their practical understanding enabling them to solve real-world challenges effectively.
- **PEO3:** To enhance students' team-building skills and leadership qualities continuously through social, cultural, and environmental activities.

Program Outcomes (POs)

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, review research literature, and analysis complex engineering problems reaching substantiated conclusions usingfirst principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-basedknowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professionalengineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation,make effective presentations, and give and receive clear instruction.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context oftechnological change.

University Syllabus

1. WAP to display “Hello World”.
2. WAP to Print additional, subtraction, multiplication, and division of two numbers.
3. WAP to swap two numbers using a third variable.
4. WAP to calculate Simple Interest.
5. WAP to find whether a given number is even or odd.
6. WAP to calculate the sum of first n natural numbers
7. WAP to design a simple calculator using Switch case.
8. WAP to display Fibonacci Series.
9. WAP to print sum of diagonal elements of a matrix.
10. WAP to swap 2 numbers by creating a function using call by value.

Course Outcomes (COs)

Upon successful completion of the course, the students will acquire:

CO1: To formulate the algorithms for simple problems and to translate given algorithms to a working and correct program

CO2: To be able to correct syntax errors as reported by the compilers

CO3: To be able to identify and correct logical errors encountered at run time

CO4: To be able to write iterative as well as recursive programs.

CO5: To be able to represent data in arrays, strings and structures and manipulate them through a program.

CO-PO Mapping

	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**
CO1				2	3			2	2	1	2	2
CO2			2		3			2	2	1	2	2
CO3				2	3			2	2	1	2	2
CO4	1			2	3			2	2	1	2	2
CO5			1	2	3			2	2	1	2	2
CO	**0.2**		**0.6**	**1.6**	**3**			**2**	**2**	**1**	**2**	**2**

CO-PSO Mapping

	PSO1	**PSO2**	**PSO3**
CO1		2	2
CO2		2	2
CO3		2	2
CO4		2	2
CO5		2	2
CO		**2**	**2**

Course Overview

Programming for Problem Solving using C is a course designed to introduce students to the fundamental concepts of computer programming using the C programming language. The course focuses on developing problem-solving skills through the application of programming techniques and algorithms.

Throughout the course, students will learn how to analyze problems, design solutions, and implement them using the C programming language.
They will gain a solid understanding of programming concepts such as variables, data types, control structures, loops, functions, and arrays.
Additionally, students will be introduced to essential algorithms and techniques used in solving common programming problems.

List of Experiments mapped with COs

DOs and DON'Ts

Dos

1. Login-on with your username and password.
2. Log off the computer every time when you leave the Lab.
3. Arrange your chair properly when you are leaving the lab.
4. Put your bags in the designated area.
5. Ask permission to print.

DON'Ts

1. Do not share your username and password.
2. Do not remove or disconnect cables or hardware parts.
3. Do not personalize the computer setting.
4. Do not run programs that continue to execute after you log off.
5. Do not download or install any programs, games or music on computer in Lab.
6. Personal Internet use chat room for Instant Messaging (IM) and Sites is strictly prohibited.
7. No Internet gaming activities allowed.
8. Tea, Coffee, Water & Eatables are not allowed in the Computer Lab.

General Safety Precautions

Precautions (In case of Injury or Electric Shock)

1. To break the victim with live electric source, use an insulator such as fire wood or plastic to break the contact. Do not touch the victim with bare hands to avoid the risk of electrifying yourself.

2. Unplug the risk of faulty equipment. If main circuit breaker is accessible, turn the circuit off.

3. If the victim is unconscious, start resuscitation immediately, use your hands to press the chest in and out to continue breathing function. Use mouth-to-mouth resuscitation if necessary.

4. Immediately call medical emergency and security. Remember! Time is critical; be best.

Precautions (In case of Fire)

1. Turn the equipment off. If power switch is not immediately accessible, take plug off.

2. If fire continues, try to curb the fire, if possible, by using the fire extinguisher or by covering it with a heavy cloth, if possible, isolate the burning equipment from the other surrounding equipment.

3. Sound the fire alarm by activating the nearest alarm switch located in the hallway.

4. Call security and emergency department immediately:

Emergency: Reception
Security: Main Gate

Guidelines to students for report preparation

All students are required to maintain a record of the experiments conducted by them. Guidelines for its preparation are as follows: -

1) All files must contain a title page followed by an index page. ***The files will not be signed by the faculty without an entry in the index page.***

2) Student's Name, Roll number and date of conduction of experiment must be written on all pages.

3) For each experiment, the record must contain the following

(i) Aim/Objective of the experiment

(ii) Pre-experiment work (as given by the faculty)

(iii) Lab assignment questions and their solutions

(iv) Test Cases (if applicable to the course)

(v) Results/ output

Note:

1. Students must bring their lab record along with them whenever they come for the lab.
2. Students must ensure that their lab record is regularly evaluated.

Lab Assessment Criteria

An estimated 10 lab classes are conducted in a semester for each lab course. These lab classes are assessed continuously. Each lab experiment is evaluated based on 5 assessment criteria as shown in following table. Assessed performance in each experiment is used to compute CO attainment as well as internal marks in the lab course.

Grading Criteria	Exemplary (4)	Competent (3)	Needs Improvement (2)	Poor (1)
AC1: Pre-Lab written work (this may be assessed through viva)	Complete procedure with underlined concept is properly written	Underlined concept is written but procedure is incomplete	Not able to write concept and procedure	Underlined concept is not clearly understood
AC2: Program Writing/ Modeling	Assigned problem is properly analyzed, correct solution designed, appropriate language constructs/ tools are applied, Program/solution written is readable	Assigned problem is properly analyzed, correct solution designed, appropriate language constructs/ tools are applied	Assigned problem is properly analyzed & correct solution designed	Assigned problem is properly analyzed
AC3: Identification & Removal of errors/ bugs	Able to identify errors/ bugs and remove them	Able to identify errors/ bugs and remove them with little bit of guidance	Is dependent totally on someone for identification of errors/ bugs and their removal	Unable to understand the reason for errors/ bugs even after they are explicitly pointed out
AC4:Execution & Demonstration	All variants of input /output are tested, Solution is well demonstrated and implemented concept is clearly explained	All variants of input /output are not tested, However, solution is well demonstrated and implemented concept is clearly explained	Only few variants of input /output are tested, Solution is well demonstrated but implemented concept is not clearly explained	Solution is not well demonstrated and implemented concept is not clearly explained
AC5:Lab Record Assessment	All assigned problems are well recorded with objective, design constructs and solution along with Performance analysis using all variants of input and output	More than 70 % of the assigned problems are well recorded with objective, design contracts and solution along with Performance analysis is done with all variants of input and output	Less than 70 % of the assigned problems are well recorded with objective, design contracts and solution along with Performance analysis is done with all variants of input and output	

LAB EXPERIMENTS

LAB EXPERIMENT 1

OBJECTIVE:

WAP to display “Hello World”.

PRE-EXPERIMENT QUESTIONS:

Q1. When and Where C-language is developed?

Q2. What is the use of printf() function in C Language?

ALGORITHIM TO DISPLAY “HELLO WORLD” IN C PROGRAM:

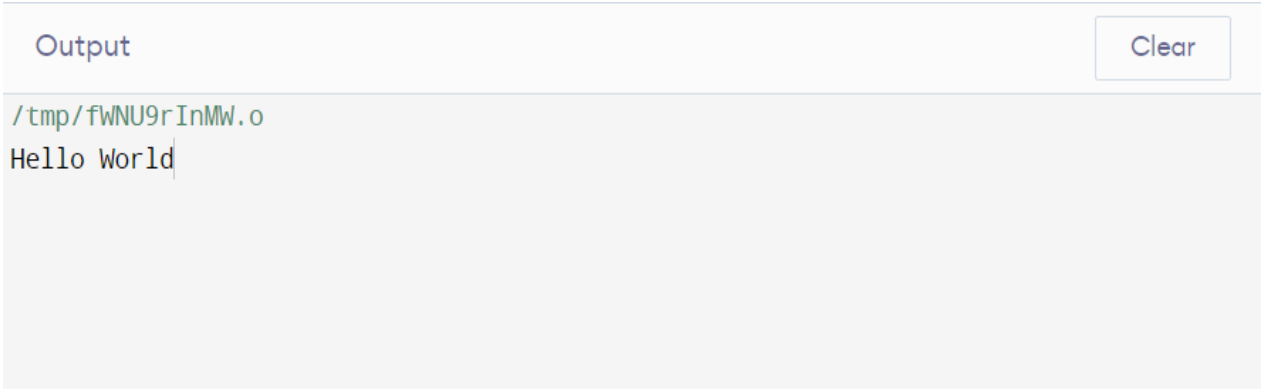
1. Start the program.
2. Include the necessary header files, such as <stdio.h>, at the beginning of the program.
3. Define the main () function with a return type of int.
4. Within the main () function, use the printf() function to output the string "Hello World" to the console. The function call should look like: printf("Hello World");.
5. Return 0 at the end of the main () function to indicate successful execution of the program.
6. End the program.

Here's an example of how the complete program would look:

PROGRAM:

```
#include  <stdio.h>
#include <conio.h>
int main()
{
        printf("Hello World");
       return 0;
 }
```

OUTPUT:



QUIZ QUESTIONS WITH ANSWERS:

Q1. What is the purpose of the #include <stdio.h> line & <conio.h> line?
Ans: The line #include <stdio.h> is a preprocessor directive in the C programming language. It is used include the standard input/output library, which provides functions for input and output operations such as reading from or writing to the console or files.

Q2. How do you save and compile a C File?
Ans: Open a text editor such as Notepad, Notepad++, or Visual Studio Code.

1. Write your C code in the text editor and save the file with a .c extension. For example, you can save it as "myprogram.c".
2. Open the Command Prompt by pressing the Windows key + R, typing "cmd", and hitting Enter.
3. In the Command Prompt, navigate to the directory where you saved your C file using the cd command. For example, if your file is saved in the "Documents" folder, you can use the command cd Documents to navigate there.
4. Once you are in the correct directory, compile the C file using a C compiler like GCC (GNU Compiler Collection) by running the command gcc -o outputfilename sourcefilename.c. For example, if your source file is "myprogram.c" and you want the output file to be named "myprogram.exe", you can use the command gcc -o myprogram.exe myprogram.c.
5. If there are no syntax errors or other issues in your code, the compiler will generate an executable file in the same directory.

You can run the compiled program by typing its name in the Command Prompt and pressing Enter. For example, in our case, you can run the program by typing myprogram.exe.

Q3. What is main()?
Ans: In programming, main() is a special function that serves as the entry point of a program. It is called automatically when the program starts execution, and it typically contains the initial code to be executed.

Q4. What is return statement?
Ans: In programming, a return statement is a statement used to terminate the execution of a function and

return a value (if applicable) back to the caller of the function. It allows a function to pass a result or outcome back to the code that called it.

Q5. What is the use of printf() statement?

Ans: The printf() statement is a function in the C programming language (and other programming languages influenced by C) that is used to print formatted output to the console or standard output. It stands for "print formatted" and is part of the standard input/output library in C.

Q6. What is #include?

Ans: In C programming, the #include directive is a preprocessor directive that is used to include external files or libraries into your C program. It allows you to use code from other files or libraries in your program by "including" them at the specified location.

Q7. What are header files?

Ans: A C header file is a text file that contains pieces of code written in the C programming language.

Q8. What are the functions included in #include<stdio.h>?

Ans: The <stdio.h> header file contains function prototypes and definitions for functions like printf(), scanf(), fprintf(), and fscanf(), which are commonly used for input and output operations. By including this header file in your C program, you make these functions available for use.

Q9. What are the functions included in #include<conio.h>?

Ans: Following are some of the functions of the header file conio.h -

1. clrscr()
2. getch()
3. getche()
4. putch()
5. cgets()
6. cputs()
7. cscanf()
8. cprintf()
9. kbhit()
10. textcolor()
11. textbackground()
12. delline
13. gotoxy
14. wherex
15. wherey

LAB EXPERIMENT 2

OBJECTIVE

WAP to Print additional, subtraction, multiplication, and division of two numbers.

PRE-EXPERIMENT QUESTIONS:

1. What are the basic data types in C?
2. What is the range of value data type variable can hold?

BRIEF DISCUSSION AND EXPLANATION:

ALGORITHIM:

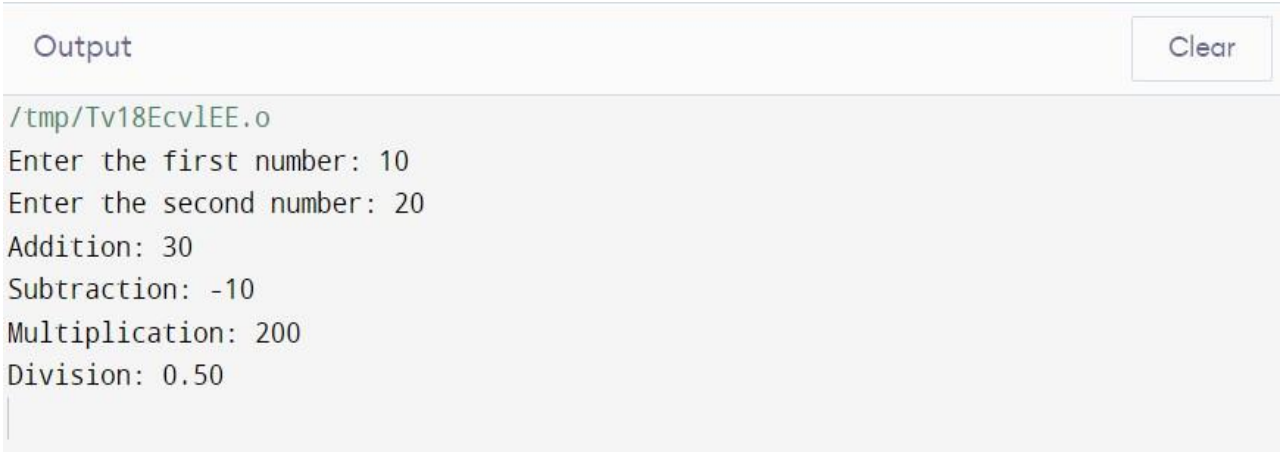
1. Start the program.
2. Declare two variables **num1** and **num2** to store the input numbers.
3. Prompt the user to enter the first number and store it in **num1**.
4. Prompt the user to enter the second number and store it in **num2**.
5. Declare four variables: **addition**, **subtraction**, **multiplication**, and **division**.
6. Perform addition by adding **num1** and **num2**, and store the result in **addition**.
7. Perform subtraction by subtracting **num2** from **num1**, and store the result in **subtraction**.
8. Perform multiplication by multiplying **num1** and **num2**, and store the result in **multiplication**.
9. Perform division by dividing **num1** by **num2**, and store the result in **division**.
10. Print the values of **addition**, **subtraction**, **multiplication**, and **division**.
11. End the program.

Here's an example of how the complete program would look:

PROGRAM:

```
#include <stdio.h>

    int main()
    {
        int num1, num2;
        int addition, subtraction, multiplication;
        float division;
        printf("Enter the first number: ");
        scanf("%d", &num1);

        printf("Enter the second number: ");
        scanf("%d", &num2);

        addition = num1 + num2;
        subtraction = num1 - num2;
        multiplication = num1 * num2;
        division = (float) num1 / num2;

        printf("Addition: %d\n", addition);
        printf("Subtraction: %d\n", subtraction);
        printf("Multiplication: %d\n", multiplication);
        printf("Division: %.2f\n", division);

    return 0;
    }
```

OUTPUT



QUIZ QUESTIONS WITH ANSWERS:

Q1. What is typecasting in C?

Ans: Type casting refers to changing an variable of one data type into another. The compiler will automatically change one type of data into another if it makes sense.

Q2. What is the difference between local and global variables?

Ans: A global variable is one that is "declared" outside of the functions in a program and can, therefore, be accessed by any of the functions. A local variable is declared inside a specific function and can only be accessed by the function in which it is declared.

Q3. What are the types of variables?

Ans: Variables are containers for storing data values, like numbers and characters.

In C, there are different **types** of variables (defined with different keywords), for example:

- int - stores integers (whole numbers), without decimals, such as 123 or -123
- float - stores floating point numbers, with decimals, such as 19.99 or -19.99
- char - stores single characters, such as 'a' or 'B'. Char values are surrounded by **single quotes**

Q4. What are format specifiers?

Ans: Format specifiers in C are used to take inputs and print the output of a type. The symbol we use in every format specifier is %. Format specifiers tell the compiler about the type of data that must be given or input and the type of data that must be printed on the screen.

Q5. What are the types of operators in C?

Ans: C has many built-in operators and can be classified into 6 types:

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Bitwise Operators
5. Assignment Operators
6. Other Operators

LAB EXPERIMENT 3

OBJECTIVE:

WAP to swap two numbers using a third variable.

PRE-EXPERIMENT QUESTIONS:

1. What are the two methods you plan to use to swap the numbers?
2. Are the numbers to be swapped integers or floating -point numbers?

BRIEF DISCUSSION AND EXPLANATION:

ALGORITHIM:

1. Start the program.
2. Declare three integer variables a, b, and temp.
3. Read the values of a and b from the user.
4. Print the original values of a and b.
5. Assign the value of a to temp.
6. Assign the value of b to a.
7. Assign the value of temp to b.
8. Print the swapped values of a and b.
9. End the program.

Here's an example of how the complete program would look:

PROGRAM:

```
#include <stdio.h>
int main()
{
        int a, b, temp;
        // Read values of a and b
        printf("Enter two numbers:\n");
        scanf("%d %d", &a, &b);
        // Print original values
        printf("Before swapping:\n");
        printf("a = %d\n", a);
        printf("b = %d\n", b);
         // Swap the numbers using a third variable
        temp = a;
        a = b;
        b = temp;
        // Print swapped values
        printf("After swapping:\n");
        printf("a = %d\n", a);
        printf("b = %d\n", b);

        return 0;
 }
```

OUTPUT

Output Clear

```
/tmp/Tv18EcvlEE.o
Enter two numbers:
10
20
Before swapping:
a = 10
b = 20
After swapping:
a = 20
b = 10
```

QUIZ QUESTIONS WITH ANSWERS:

Q1. What is the precedence of arithmetic operators?
Ans: The precedence order (from highest to lowest) of arithmetic operators is %, *, /, +, –

Q2. What is the use of modulus operator?
Ans: It divides the given numerator by the denominator to find a result.

Q3. Are the numbers to be swapped integers or floating -point numbers?
Ans: They are integers.

Q4. What is the time complexity to swap two numbers?
Ans: It is constant.

LAB EXPERIMENT 4

OBJECTIVE:

WAP to calculate Simple Interest.

PRE-EXPERIMENT QUESTIONS:

1. What is the formula of Simple Interest?
2. What do you mean by ampersand in c?

BRIEF DISCUSSION AND EXPLANATION:

Let's understand the logic of writing the simple interest program in c. For calculating simple interest, we must have the values of principal, rate, and the total time of interest.

ALGORITHIM:

Let's understand the algorithm to write the simple interest program in C.

1. Store the values of the Principal, Time, and the Rate in the particular data types.
2. Now use this formula (P * R * T) / 100 to calculate the simple interest. The P is the principal, R is the rate of interest, and the T is the total period of time.
3. The calculated value from the above expression is the Simple interest.

Here's an example of how the complete program would look:

PROGRAM:

```
#include <stdio.h>
int main()
	{
		float principal, rate, time, interest;
		printf("Enter the principal amount: ");
		scanf("%f", &principal);

		printf("Enter the rate of interest: ");
		scanf("%f", &rate);

		printf("Enter the time period (in years): ");
		scanf("%f", &time);

		// Calculate the simple interest
		interest = (principal * rate * time) / 100;

		printf("Simple Interest = %.2f\n", interest);

	return 0;
	}
```

OUTPUT

```
Output                                                    Clear
/tmp/Tv18EcvlEE.o
Enter the principal amount: 1000
Enter the rate of interest: 10
Enter the time period (in years): 5
Simple Interest = 500.00
```

QUIZ QUESTIONS WITH ANSWERS:

Q1. What is the time complexity of finding the simple interest in the C language?
Ans: It is constant

Q2. What is the size of Float operator?
Ans: The size of the **float** data type in most programming languages is typically 32 bits, or 4 bytes.

Q3. What is the range of floating-point number?
Ans: Minimum normalized positive value: 1.17549435 × 10^(-38). Maximum finite value: 3.40282347 × 10^38

Q4. What is the formula to find simple interest?
Ans: Simple Interest = (Principal * Rate * Time)/10

LAB EXPERIMENT 5

OBJECTIVE:

WAP to find whether a given number is even or odd.

PRE-EXPERIMENT QUESTIONS:

1. What is the purpose of the if-else statement in the program?
2. Draw a Flowchart of this program?

BRIEF DISCUSSION AND EXPLANATION:

Odd and even numbers are types of integers, which are whole numbers (positive, negative, or zero) that are not fractions or decimals.
An even number is any integer that is divisible by 2, meaning it has no remainder when divided by 2. For example, 2, 4, 6, 8, and 10 are all even numbers.
On the other hand, an odd number is any integer that is not divisible by 2, meaning it has a remainder of 1 when divided by 2.
For example, 1, 3, 5, 7, and 9 are all odd numbers.

ALGORITHIM:

1. Start the program.
2. Read a number from the user and store it in a variable number.
3. Check if number modulo 2 is equal to 0.
4. If the result is 0, the number is even. Print a message indicating that the number is even.
1. If the result is not 0, the number is odd. Print a message indicating that the number is odd.
2. End the program.

Here's an example of how the complete program would look:

PROGRAM:

```
#include <stdio.h>
int main()
{
	int number;
	printf("Enter a number: ");
	scanf("%d", &number);
	if (number % 2 == 0)
	{
		printf("%d is an even number.\n", number);
	}
	else
	{
		printf("%d is an odd number.\n", number);
	}

	return 0;
}
```

OUTPUT

```
Output                                                    Clear
/tmp/Tv18EcvlEE.o
Enter a number: 21
21 is an odd number.
```

QUIZ QUESTIONS WITH ANSWERS:

Q1. Can the program check whether a decimal number is odd or even?
Ans: Yes, a program can check whether a decimal number is odd or even.

Q2. Can the program be modified to perform other operations on the input number?
Ans: Yes, we can modify.

Q3. What is if-else statement?
Ans: In C, the **if-else** statement is a conditional statement used to control the flow of a program based on a condition. It allows you to specify a block of code to be executed if a certain condition is true, and an alternative block of code to be executed if the condition is false.

Q4. What is nested if-else?
Ans: Nested **if-else** statements refer to the usage of an **if-else** statement inside another **if** or **else** block. This allows for more complex conditional branching and the ability to handle multiple conditions.

Q5. What is else-if ladder?
Ans: An "else if ladder" is a term used to describe a sequence of multiple **else if** statements following an initial **if** statement. It allows for testing multiple conditions in a structured manner and executing different blocks of code based on the outcome of those conditions.

Q6. What is the difference between nested if else and else if ladder?
Ans: The main difference between nested **if-else** statements and an "else if ladder" lies in their structure and the way they handle multiple conditions.
Nested **if-else** statements involve placing one **if-else** statement inside another **if** or **else** block. This allows for a more hierarchical structure, where each nested **if-else** statement is evaluated only if its outer condition is true. This structure allows for more complex branching logic and the ability to handle multiple conditions with different levels of nesting.
On the other hand, an "else if ladder" refers to a sequence of multiple **else if** statements following an initial **if** statement. This structure provides a more linear and straightforward way of testing multiple conditions one by one. Each **else if** condition is evaluated only if the preceding conditions are false, and the associated block of code is executed if the condition is true. If none of the **else if** conditions are true, the final **else** block is executed.

LAB EXPERIMENT 6

OBJECTIVE:

WAP to calculate the sum of first n natural numbers.

PRE-EXPERIMENT QUESTIONS:

1. What is time complexity and **space complexity?**
2. What is the significance of the loops?

BRIEF DISCUSSION AND EXPLANATION:

ALGORITHIM:

Here's a step-by-step breakdown of the algorithm:

We start by reading the value of n, which represents the number of natural numbers we want to sum.
We initialize a variable sum to 0, which will store the sum of the numbers.
We also initialize a variable i to 1, which will be used as a counter to iterate through the numbers.
Using a loop, we iterate from 1 to n. In each iteration, we add the value of i to the sum and increment i by 1.
After the loop ends, we print the value of the sum, which represents
the sum of the first n natural numbers.
Finally, the program ends.

Here's an example of how the complete program would look:

PROGRAM:

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

   printf("Enter a positive integer n: ");
   scanf("%d", &n);

   // Calculate the sum of the first n natural numbers
   for (int i = 1; i <= n; i++)
        {
              sum += i;
        }

   printf("The sum of the first %d natural numbers is: %d\n", n, sum);

   return 0;
}
```

OUTPUT

```
Output                                                    Clear
/tmp/Tv18EcvlEE.o
Enter a positive integer n: 10
The sum of the first 10 natural numbers is: 55
```

QUIZ QUESTIONS WITH ANSWERS:

Q1. What are the jumping statements in C Language?

Ans: In the C programming language, there are three types of jumping statements:

1. **break**: The **break** statement is used to terminate the execution of a loop (**for**, **while**, or **do-while**) or a **switch** statement. When a **break** statement is encountered, the program flow immediately exits the innermost loop or **switch** statement, and control is transferred to the next statement after the loop or **switch** block.
2. **continue**: The **continue** statement is used to skip the remaining statements within a loop and move the control to the next iteration of the loop. When a **continue** statement is encountered, the program flow jumps to the loop's increment or update statement (e.g., the increment expression in a **for** loop or the

update expression in a **while** or **do-while** loop), bypassing any remaining statements within the loop block.

3. **goto**: The **goto** statement allows you to transfer control to a labeled statement within the same function. It is considered a "jump" statement as it can transfer the control flow to any labeled statement, including those located outside the current loop or block. However, the use of **goto** is generally discouraged due to its potential to create complex and hard-to-maintain code. It can make the programflow difficult to follow and can lead to code that is hard to debug.

Q2. What are loops?
Ans: In the C programming language, loops are control structures that allow you to repeat a block of code multiple times.

Q3. Which loop is good for programming?
Ans: In general, the choice of loop depends on the specific requirements and logic of your program. It is recommended to choose the loop that best suits the task at hand, provides clarity and readability, and ensures the correct behavior of the program. It's also important to consider factors such as code efficiency, maintainability, and readability when making your decision.

Q4. What are the types of loops?
Ans: There are three types of loops commonly used in C:

1. **for** loop: The **for** loop is used when you know the number of iterations in advance. It consists of three parts: initialization, condition, and increment/decrement.

The initialization is executed once at the beginning, the condition is checked before each iteration, and the increment/decrement is executed after each iteration. The loop continues as long as the condition is true.

2. **while** loop: The **while** loop is used when you want to repeat a block of code based on a certain condition. It checks the condition before each iteration.

The loop will continue as long as the condition is true. If the condition is false initially, the loop is never executed.

3. **do-while** loop: The **do-while** loop is similar to the **while** loop, but it checks the condition at the end of each iteration. This guarantees that the loop will be executed at least once, even if the condition is false.

   The loop will continue as long as the condition is true. The code block is executed first, and then the condition is checked. If the condition is false, the loop is exited.

Q5. What is the syntax of for loop?
Ans: The syntax of a **for** loop is as follows:

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

LAB EXPERIMENT 7

OBJECTIVE:

WAP to design a simple calculator using Switch case.

PRE-EXPERIMENT QUESTIONS:

1. What is the Syntax of switch Statement in C?
2. What is the purpose of control statements in C?

BRIEF DISCUSSION AND EXPLANATION:

ALGORITHIM:

1. Start the program.
2. Display a calculator menu with various operations and
their corresponding numbers.
3. Prompt the user to enter their choice of operation.
4. Read the user's choice and store it in a variable choice.
5. Prompt the user to enter two numbers for the operation.
6. Read the two numbers and store them in variables num1 and num2.
7. Use a switch case statement with choice as the controlling expression.
8. For each case, perform the corresponding operation based on the user's choice.
- If choice is 1, perform addition and display the result.
- If choice is 2, perform subtraction and display the result.
- If choice is 3, perform multiplication and display the result.
- If choice is 4, perform division and display the result.
- If choice is not 1, 2, 3, or 4, display an error message for an invalid choice.
9. End the switch case.
10. End the program.

Here's an example of how the complete program would look:

PROGRAM:

```
#include <stdio.h>
int main()
{
   int choice;
   float num1, num2, result;
   printf("Calculator Menu:\n");
   printf("1. Addition\n");
   printf("2. Subtraction\n");
   printf("3. Multiplication\n");
   printf("4. Division\n");
   printf("Enter your choice (1-4): ");
   scanf("%d", &choice);
   printf("Enter two numbers: ");
   scanf("%f %f", &num1, &num2);
   switch (choice) {
      case 1:
         result = num1 + num2;
         printf("Result: %.2f\n", result);
         break;
      case 2:
         result = num1 - num2;
         printf("Result: %.2f\n", result);
         break;
      case 3:
         result = num1 * num2;
         printf("Result: %.2f\n", result);
         break;
      case 4:
         if (num2 != 0) {
            result = num1 / num2;
            printf("Result: %.2f\n", result);
         } else {
            printf("Error: Division by zero is not allowed.\n");
         }
         break;
      default:
         printf("Invalid choice.\n");
         break;
   }
```

```
    return 0;
}
```

OUTPUT:

QUIZ QUESTIONS WITH ANSWERS:

Q1. What is switch case?
Ans: In the C programming language, the switch case statement is a control structure used for multiple conditional branching. It allows you to select one of many code blocks to execute based on the value of a single expression or variable.

Q2. How switch case works?
Ans: Here's how it works:

1. The **expression** is evaluated once.
2. The value of the **expression** is compared with each **case** value.
3. If a match is found, the corresponding code block following that **case** is executed.
4. The **break** statement is used to exit the **switch** block. If no **break** is present, execution will continue to the next **case** without any further comparisons.
5. If the **expression** doesn't match any **case** value, the code block following the **default** label (if present) is executed.
6. After executing the corresponding code block, control usually continues with the statement immediately following the **switch** block.

Q3. How switch case is different from loops?
Ans: Switch case and loops are both control structures in programming, but they serve different purposes and operate

in different ways.

1. Purpose:
    - Switch case: The switch case statement is used for conditional branching based on the value of a single expression or variable. It allows you to select one of multiple code blocks to execute based on different cases.
    - Loops: Loops are used for repetitive execution of a block of code. They allow you to repeat a set of instructions multiple times until a certain condition is met.
2. Execution Flow:
    - Switch case: In a switch case statement, the expression is evaluated once, and based on its value, control jumps to the corresponding case label. Once the code block associated with that case is executed, control usually continues with the statement immediately following the switch case statement.
    - Loops: In loops, a block of code is repeated until a specific condition is satisfied. The code within the loop is executed repeatedly, and the loop continues until the condition becomes false. After the loop terminates, control continues with the next statement after the loop.
3. Usage:
    - Switch case: It is used for selecting among multiple options based on the value of a single expression. For example, you can use a switch case statement to implement a menu system where different actions are taken based on user input.
    - Loops: Loops are used when you want to perform a specific set of instructions repeatedly, such as iterating over a collection of data, performing calculations, or executing a block of code a fixed number of times.

In summary, the switch case statement is used for selective execution of code based on different cases, whereas loops are used for repetitive execution of code until a certain condition is met.

LAB EXPERIMENT 8

OBJECTIVE:

WAP to display Fibonacci Series.

PRE-EXPERIMENT QUESTIONS:

1. What is flowchart of for loop?
2. How can I terminate a "for" loop prematurely?

BRIEF DISCUSSION AND EXPLANATION:

Fibonacci Series generates subsequent number by adding two previous numbers. Fibonacci series starts from two numbers − F0 & F1. The initial values of F0 & F1 can be taken 0, 1 or 1, 1 respectively.

ALGORITHIM:

1. Start the program.
2. Prompt the user to enter the number of terms they want in the Fibonacci series and store it in a variable n.
3. Initialize two variables, term1 and term2, with the first two terms of the series, which are 0 and 1, respectively.
4. Print the value of term1 as the first term of the series.
5. Start a loop from 2 to n.
6. Within the loop, calculate the next term of the series by adding term1 and term2 and store it in a variable next Term.
7. Print the value of next Term.
8. Update the values of term1 and term2 for the next iteration by assigning term2 to term1 and next Term to term2.
9. End the loop.
10. End the program.

Here's an example of how the complete program might look:

PROGRAM:

```
#include <stdio.h>

int main()
{
        int n, i, term1 = 0, term2 = 1, nextTerm;
        printf("Enter the number of terms: ");
        scanf("%d", &n);
        printf("Fibonacci Series: ");
        for (i = 1; i <= n; i++)
        {
        printf("%d ", term1);
        nextTerm = term1 + term2;
        term1 = term2;
        term2 = nextTerm;
        }
        printf("\n");

   return 0;
}
```

OUTPUT:

QUIZ QUESTIONS WITH ANSWERS:

Q1. How will you differentiate ++a and a++?

Ans: The expressions **++a** and **a++** are both increment operators used in programming languages like C. However, they differ in terms of when the increment operation is applied to the variable and the value they evaluate to.

1. **++a** (Prefix Increment):
    - The **++a** operator is known as the prefix increment operator.
    - When used as a prefix, the increment operation is applied to the variable **a** before the value of **a** is evaluated or used in any other expressions.
    - The value of **++a** is the incremented value of **a** after the increment operation.

- For example, if **a** is initially 5, **++a** will increment **a** to 6 and evaluate to 6.

2. **a++** (Postfix Increment):
   - The **a++** operator is known as the postfix increment operator.
   - When used as a postfix, the increment operation is applied to the variable **a** after the value of **a** is evaluated or used in other expressions.
   - The value of **a++** is the original value of **a** before the increment operation.
   - After the value is evaluated, the variable **a** is then incremented.
   - For example, if **a** is initially 5, **a++** will evaluate to 5 and then increment **a** to 6.

Q2. What are loops?
Ans: In the C programming language, loops are control structures that allow you to repeat a block of code multiple times.

Q3. Which loop is good for programming?
Ans: In general, the choice of loop depends on the specific requirements and logic of your program. It is recommended to choose the loop that best suits the task at hand, provides clarity and readability, and ensures the correct behavior of the program. It's also important to consider factors such as code efficiency, maintainability, and readability when making your decision.

Q4. What are the types of loops?
Ans: There are three types of loops commonly used in C:

1. **for** loop: The **for** loop is used when you know the number of iterations in advance. It consists of three parts: initialization, condition, and increment/decrement.

The initialization is executed once at the beginning, the condition is checked before each iteration, and the increment/decrement is executed after each iteration. The loop continues as long as the condition is true.

2. **while** loop: The **while** loop is used when you want to repeat a block of code based on a certain condition. It checks the condition before each iteration.

The loop will continue as long as the condition is true. If the condition is false initially, the loop is never executed.

3. **do-while** loop: The **do-while** loop is similar to the **while** loop, but it checks the condition at the end of each iteration. This guarantees that the loop will be executed at least once, even if the condition is false.

   The loop will continue as long as the condition is true. The code block is executed first, and then the condition is checked. If the condition is false, the loop is exited.

Q5. What is the syntax of for loop?
Ans: The syntax of a **for** loop is as follows:

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

Q6. Can we write Fibonacci Series program in while loop?

Ans: Yes, you can write a program to generate the Fibonacci series using a while loop. The Fibonacci series is a sequence of numbers in which each number is the sum of the two preceding ones.

LAB EXPERIMENT 9

OBJECTIVE:

WAP to print sum of diagonal elements of a matrix.

PRE-EXPERIMENT QUESTIONS:

1. What do you understand by concept of Array?
2. How to pass an array 1D and 2D as parameter in C?

BRIEF DISCUSSION AND EXPLANATION:

In this program, the matrix is defined as a 2D array of size 3x3. You can modify the matrix values and dimensions as needed.

Algorithm to find sum of diagonal matrix:

1. Start
2. Initialize a variable diagonal sum to 0.
3. Read the dimensions of the matrix (number of rows and columns).
4. Check if the matrix is square (number of rows = number of columns). If not, display an error message and end the algorithm.
5. Read the elements of the matrix into a 2D array.
6. Set a loop variable i to 0.
7. Repeat steps 8 to 9 while i is less than the number of rows.
8. Add the value of the element at index i, i to the diagonal sum.
9. Increment i by 1.
10. Display the value of diagonal sum as the sum of the diagonal elements of the matrix.
11. End.

PROGRAM:

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

	int i, j, m = 3, n = 3, a = 0, sum = 0;

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

	// if both rows and columns are equal then it is
	// possible to calculate diagonal sum
	if (m == n) {

		// printing the input matrix
		printf("The matrix is \n");

		// iterates number of rows
		for (i = 0; i < m; ++i) {

			// iterates number of columns
			for (j = 0; j < n; ++j) {
				printf(" %d", matrix[i][j]);
			}
			printf("\n");
		}

	for (i = 0; i < m; ++i) {

			// calculating the main diagonal sum
			sum = sum + matrix[i][i];

			// calculating the off-diagonal sum
			a = a + matrix[i][m - i - 1];
		}

		// printing the result
		printf("\nMain diagonal elements sum is = %d\n", sum);
```

```
            printf("Off-diagonal elements sum is = %d\n", a);
        }
        else
            // if both rows and columns are not equal then it is
            // not possible to calculate the sum
            printf("not a square matrix\n");
        return 0;
}
```

OUTPUT:

```
Output                                                    Clear
/tmp/K0D2aCQGug.o
The matrix is
 1 2 3
 4 5 6
 7 8 9

Main diagonal elements sum is = 15
Off-diagonal elements sum is = 15
```

QUIZ QUESTIONS WITH ANSWERS:

Q1. What is the logic used to add elements of main diagonals?

Ans: Here's an example of the logic to add the elements of the main diagonal:

1. Initialize a variable, let's say **sum**, to 0. This variable will hold the sum of the main diagonal elements.
2. Iterate over the rows and columns of the matrix.
3. For each element at index **(i, j)**:
    - If **i** is equal to **j**, it means the element is on the main diagonal. Add it to the **sum** variable.
4. After the loop finishes, the **sum** variable will contain the sum of the main diagonal elements.

Q2. Can we declare an array size as a negative number in C language?

Ans: In the C programming language, it is not possible to declare an array with a negative size. The size of an array must be a non-negative constant value or an expression that evaluates to a non-negative value. Declaring an array with a negative size will result in a compilation error

LAB EXPERIMENT 10

OBJECTIVE:

WAP to swap 2 numbers by creating a function using call by value.

PRE-EXPERIMENT QUESTIONS:

1. How do you define a function in C?
2. What is the notation for following functions?

BRIEF DISCUSSION AND EXPLANATION:

ALGORITHIM:

Here's the algorithm in C to swap two numbers using a function with call by value:

1. Start the program.
2. Define a function swap numbers that takes two integer parameters a and b.
3. Inside the function:
4. Declare a temporary variable temp of type integer.
5. Assign the value of a to temp.
6. Assign the value of b to a.
7. Assign the value of temp to b.
8. In the main function:
9. Declare two integer variables, x and y, and initialize them with the numbers to be swapped.
10. Print the values of x and y before swapping.
11. Call the swap numbers function, passing x and y as arguments.
12. Print the values of x and y after swapping.
13. End the program.

PROGRAM:

```
#include <stdio.h>

void swap(int, int);

int main()
{
  int x, y;

printf("Enter the value of x and y\n");
  scanf("%d%d",&x,&y);

  printf("Before Swapping\nx = %d\ny = %d\n", x, y);

  swap(x, y);

  printf("After Swapping\nx = %d\ny = %d\n", x, y);

  return 0;
}
void swap(int a, int b)
{
  int temp;

  temp = b;
  b = a;
  a = temp;
   printf("Values of a and b is %d  %d\n",a,b);
}
```

OUTPUT

Output Clear

```
/tmp/K0D2aCQGug.o
Enter the value of x and y
10
20
Before Swapping
x = 10
y = 20
Values of a and b is 20  10
After Swapping
x = 10
y = 20
```

QUIZ QUESTIONS WITH ANSWERS:

Q1. How are parameters passed by value in C?
Ans: In C, function parameters are typically passed by value. This means that when you call a function and pass arguments to its parameters, a copy of the argument's value is made and assigned to the corresponding parameter within the function. Any modifications made to the parameter within the function do not affect the original argument outside the function.

Q2. Can I modify the original values of variables in call by value?
Ans: No, in the "call by value" parameter passing mechanism in C, you cannot directly modify the original values of variables passed as function arguments. When a variable is passed by value, a copy of its value is made and assigned to the corresponding parameter within the function. Any modifications made to the parameter within the function only affect the parameter itself, not the original variable outside the function.

Q3. What is the difference between call by value and call by reference?
Ans: The main difference between "call by value" and "call by reference" lies in how function parameters are passed and how they affect the original values of variables outside the function.

1. Call by Value:
    - In "call by value," a copy of the value of the argument is made and assigned to the corresponding parameter within the function.
    - Any modifications made to the parameter within the function do not affect the original argument outside the function.
    - The original variable remains unchanged.
    - Call by value is the default parameter passing mechanism in C.
    - It is used for passing basic data types (integers, floats, characters) and small structures.
2. Call by Reference:
    - In "call by reference," the address (reference) of the argument is passed to the corresponding parameter within the function.
    - By using pointers or references, you can access and modify the original variable directly.
    - Modifications to the parameter within the function affect the original argument outside the function.
    - The original variable can be changed within the function.
    - Call by reference is used when you want to modify the original variable or pass large data structures efficiently without making copies.
    - In C, call by reference is achieved by passing pointers to variables as function arguments.

This lab manual has been updated by

Prof. Sakshi Ahuja
(Sakshi.ahuja@ggnindia.dronacharya.info)

Crosschecked By
HOD Applied Sciences and Humanities

Please spare some time to provide your valuable feedback.