PABNA UNIVERSITY OF SCIENCE AND TECHNOLOGY

Faculty of Engineering and Technology

Department of Information and Communication Engineering

Lab Report

Course Title: **System Analysis and Software Testing Sessional**

Course Code: **ICE_4204**

Submitted by	Submitted to
ALAMIN Roll No: 200610 Reg. No: 1065375 Session: 2019-2020. 4^{th} year 2^{nd} semester, Department of Information and Communication Engineering, Pabna University of Science and Technology.	Md. Anwar Hossain Professor, Department of Information and Communication Engineering, Pabna University of Science and Technology.

Submission Date: 21/05/2025

....

Signature

Index

Program No: 01
Program Name: Write a program in python to develop a simple calculator that can take a number, an operator (addition/ subtraction/ multiplication/ division/ modulo) and another number consecutively as input and the program will display the output after pressing the "=" sign.

Sample input: 1+2=, 8%4= ;
Sample output: 1+2=3, 8%4=0.

Objectives:

- To develop a simple calculator that takes an arithmetic expression in the format number operator number = and computes the result.
- To implement basic arithmetic operations (addition, subtraction, multiplication, division, modulo) using conditional logic in Python.

Theory: A simple calculator program is a basic software application that performs fundamental arithmetic operations such as addition (+), subtraction (-), multiplication (*), division (/), and modulo (%). In this program, the user provides input in the form of an expression like 1+2= or 8%4=, where the program reads the input as a string, parses the operands and operator, and performs the respective mathematical operation.

The program uses conditional statements (if, elif) to detect which operator is used and then applies the appropriate operation on the operands. It also includes basic error handling to manage invalid inputs or operations like division or modulo by zero. This type of program helps learners understand user input handling, string manipulation, and control structures in Python.

Algorithm: Simple Calculator

1. **Start**
2. **Prompt user** to input an arithmetic expression in the form: number operator number = (e.g., 3+5=).
3. **Read** the input expression as a string and **remove leading/trailing spaces**.
4. **Check** if the expression ends with the '=' sign:

- **If yes**:
    1. **Remove** the '=' from the end of the expression.
    2. **Try** to:
        - **Evaluate** the expression using the eval() function.
        - **Print** the original expression along with the result (e.g., 3+5=8).
    3. **Catch exceptions** (like syntax errors or invalid operations):
        - **Print** an error message indicating the evaluation issue.
- **If no**:
    - **Print** a message stating: "Expression must end with '='."

5. **End**

Python Code:

```python
def simple_calculator():
    expression = input("Enter expression (e.g., 4*8=): ").strip()
    # Check if input ends with '='
    if expression.endswith('='):
        expression = expression[:-1]  # remove the '=' sign
        try:
            result = eval(expression)
            print(f"{expression}={result}")
        except Exception as e:
            print(f"Error evaluating expression: {e}")
    else:
        print("Expression must end with '='.")
# Run the calculator
simple_calculator()
```

Input:

```
34-27=
54%5=
```

Output:

```
34-27=7
54%5=4
```

Program No: 02
Program Name: Write a program in python that will take two 'n' integers as input until a particular operator and produce 'n' output.

Sample input: 4 5 7 8 20 40 +;
Sample output: 9 15 60.

Objectives:

- To develop a Python program that performs arithmetic operations in pairwise fashion on a sequence of integers based on a specified operator.
- To practice list handling, loops, and condition-based execution for grouped data processing.

Theory: This program performs arithmetic operations on pairs of integers provided by the user. The input consists of a sequence of even-numbered integers followed by a single arithmetic operator (+, -, *, /, or %). The program reads the input, separates the numbers and the operator, and then processes the integers two at a time (pairwise). For each pair, it performs the specified operation and stores the result.

For example, if the input is 4 5 7 8 20 40 +, the program takes the pairs (4,5), (7,8), and (20,40), and applies addition to each pair to produce the outputs 9, 15, and 60 respectively. This approach demonstrates key concepts in Python such as input parsing, list operations, loops, conditionals, and basic arithmetic processing. It also helps learners understand how to structure data-driven logic using control structures.

Algorithm: Pairwise Operation Calculator

```
1. Start.
2. Take input: numbers followed by an operator (e.g., 4 5 6 7 +).
3. Split the input into a list of tokens.
4. Extract the last token as the operator.
5. Convert the rest into a list of integers.
6. If the number of integers is odd, print an error and stop.
7. For each pair of numbers:
      o Apply the operator (+, -, *, /, %).
      o Store the result.
8. Print all results.
9. End.
```

Python Code:

```
def pairwise_operation():
    # Take the full input line
     input_str = input("Enter numbers followed by an operator (e.g., 4 5 7 8 20 40 +): ").strip()
    # Split into parts
    tokens = input_str.split()

    if len(tokens) < 3:
        print("Not enough input.")
        return

    operator = tokens[-1]  # operator is the last item
    numbers = list(map(int, tokens[:-1]))  # all before operator are numbers

    if len(numbers) % 2 != 0:
        print("Odd number of inputs; one number has no pair.")
        return
    results = []
    for i in range(0, len(numbers), 2):
        a, b = numbers[i], numbers[i+1]
        if operator == '+':
            results.append(a + b)
        elif operator == '-':
            results.append(a - b)
        elif operator == '*':
            results.append(a * b)
        elif operator == '/':
            results.append(a / b if b != 0 else 'Error')
        elif operator == '%':
            results.append(a % b if b != 0 else 'Error')
        else:
            print("Unsupported operator.")
            return
    print("Output:", ' '.join(str(r) for r in results))
# Run the function
pairwise_operation()
```

Input:

```
Enter numbers followed by an operator:
34 56 23 98 *
```

Output:

```
Output: 1904 2254
```

Program No: 03
Program Name: Write a program in python to check whether a number or string is palindrome or not.
N.B.: Your program must not take any test case number such as 1 or 2 for the desired cases from the user. The program should directly take the number or string as input and produce the correct result.
Objective:

- To develop a Python program that checks whether a user-inputted string or number is a palindrome without requiring any test case count input.

Theory: A palindrome is a word, phrase, number, or other sequence of characters that reads the same forward and backward, ignoring spaces, punctuation, and capitalization. Examples include words like "madam" or numbers like 12321.
In this Python program, the user is prompted to enter a string or number. The input is converted into a string to handle both text and numeric cases uniformly. The program then checks whether the input is equal to its reverse by using Python's slicing technique (value[::-1]). If they match, the input is identified as a palindrome; otherwise, it is not.
This program demonstrates fundamental concepts such as string manipulation, user input handling, and conditional logic. It is useful for understanding how to work with sequences and conditions in Python programming.
Algorithm: Palindrome Checker

1. **Start**
2. **Input** a string or number from the user.
3. **Remove extra spaces** and convert input to **lowercase**.
4. **Reverse** the normalized input.
5. **Compare** the original input with the reversed one:
    - If both are the same, **print** it is a palindrome.
    - Otherwise, **print** it is not a palindrome.
6. **End**

Python Code:

```
def check_palindrome():
    user_input = input("Enter a number or string: ").strip()
    # Normalize input (optional: lowercase for case-insensitive check)
    normalized = user_input.lower()
    # Check if it is a palindrome
    if normalized == normalized[::-1]:
        print(f"{user_input} is a palindrome.")
    else:
        print(f"{user_input} is not a palindrome.")

# Run the function
check_palindrome()
```

Input:

```
Enter a number or string: madam
```

Output:

```
madam is a palindrome.
```

Program No: 04
Program Name: Write down the ATM system specifications and report the various bugs.
Solution:
ATM System Specifications
1. Functional Specifications

- User Authentication
    - Card insertion (or cardless via OTP)
    - PIN entry and validation
- Account Services
    - Balance inquiry
    - Cash withdrawal
    - Cash deposit (if supported)
    - Mini-statement generation
    - Fund transfer between accounts
- Transaction Services
    - Validate sufficient balance
    - Dispense cash in correct denominations
    - Update account balance after each transaction
    - Print receipts (optional)
- Security Features
    - PIN encryption and timeout
    - Daily transaction limit
    - Card blocking after 3 incorrect PIN attempts
- Network Communication
    - Connects to bank server to verify and process transactions

Common Bugs in ATM Systems

Bug Type	Description
1. Card Read Error	ATM fails to read card due to sensor malfunction or software timeout.
2. PIN Validation Bug	Valid PIN rejected or invalid PIN accepted due to logic error.
3. Cash Dispensing	Amount requested differs from amount dispensed (e.g.,

Bug Type	Description
Error	incorrect denominations).
4. Balance Update Bug	Account not updated correctly after transaction (e.g., withdrawal without balance deduction).
5. Session Timeout Bug	Session doesn't timeout after inactivity or times out too early.
6. UI Bugs	Display errors (e.g., misaligned text, incorrect prompts).
7. Duplicate Transactions	Repeated transaction due to system lag or button press.
8. Receipt Printing Error	Missing or incorrect receipt information.
9. Security Flaws	No encryption in PIN transmission, or failure to block card after multiple wrong PINs.
10. Network Error Handling	Crashes or freezes when connection to bank server fails.

Sample Bug Report (Example)
Bug ID: ATM-004
Title: Withdrawal succeeds but balance not updated
Severity: High
Description: After withdrawing 5000 BDT, balance remains the same. Occurs 2/5 times during testing.
Steps to Reproduce:
1. Insert card
2. Enter PIN
3. Withdraw 5000 BDT
4. Check balance
Expected: Balance should reduce by 5000 BDT
Actual: Balance remains unchanged

Program No: 05
Program Name: Write a program in python to find out the factorial of a number using while or for loop. Also verify the results obtained from each case.
Objectives:

- Understand how to compute factorial using iterative methods.
- Learn to implement both for and while loop constructs.
- Verify correctness by comparing outputs.

Theory: The factorial of a number is a basic concept in mathematics and computer science, commonly used in areas such as combinatorics, algebra, and probability.

The factorial of a non-negative integer n, denoted by n!, is the product of all positive integers less than or equal to n. It is defined as:

$$n! = n \times (n-1) \times (n-2) \times \cdots \times 2 \times 1$$

For example:

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$
$$0! = 1 (\text{by definition})$$

Python Code:

```
def factorial_for_loop(n):
    fact = 1
    for i in range(1, n + 1):
        fact *= i
    return fact

def factorial_while_loop(n):
    fact = 1
    i = 1
    while i <= n:
        fact *= i
        i += 1
    return fact

# Take input from the user
num = int(input("Enter a non-negative integer: "))
```

```
if num < 0:
    print("Factorial is not defined for negative numbers.")
else:
    fact_for = factorial_for_loop(num)
    fact_while = factorial_while_loop(num)

    print(f"Factorial of {num} using for loop: {fact_for}")
    print(f"Factorial of {num} using while loop: {fact_while}")

    # Verifying both results
    if fact_for == fact_while:
        print("Both methods give the same result.")
    else:
        print("Mismatch found between methods.")
```

Input:

```
Enter a non-negative integer:10
```

Output:

```
Factorial of 10 using for loop: 3628800
Factorial of 10 using while loop: 3628800
 Both methods give the same result.
```

Program No: 06
Program Name: Write a program in python that will find sum and average of array using do while loop and 2 user inputs.
Objectives:

- To accept two numeric inputs from the user as elements of an array.
- To compute the sum and average of these elements using a simulated do-while loop.
- To practice input handling, loop control, and basic arithmetic operations in Python.

Theory: An array is a data structure that stores a collection of items (values or variables), typically of the same data type. In Python, lists are used as dynamic arrays. In many programming languages, a do-while loop executes a block of code at least once, and then continues looping while a condition is true.

Python does not have a built-in do-while loop, but it can be simulated using a while True: loop with a break statement based on a condition.

Sum and Average:

- Sum of an array: The result of adding all the elements in the array.
- Average of an array: The sum of the elements divided by the total number of elements.

$$\text{Sum} = a_1 + a_1 + \cdots + a_n$$

$$\text{Average} = \frac{Sum}{n}$$

Algorithm: Factorial Using For and While Loop

1. Start
2. Take a number n as input
3. If n is negative, show an error and stop
4. Use a **for loop** to calculate factorial and store in fact_for
5. Use a **while loop** to calculate factorial and store in fact_while
6. Print both results
7. If both results are same, print **"Both methods give the same result"**
8. Else, print **"Mismatch found"**
9. End

Python Code:

```
arr = list(map(float, input("Enter 2 numbers separated by space: ").split()))
# Make sure exactly two numbers are entered
if len(arr) != 2:
    print("Please enter exactly two numbers.")
else:
    index = 0
    sum_of_elements = 0

    # Simulated do-while loop
    while True:
        sum_of_elements += arr[index]
        index += 1
        if index >= len(arr):
            break

    average = sum_of_elements / len(arr)
    print(f"\nArray elements: {arr}")
    print(f"Sum: {sum_of_elements}")
    print(f"Average: {average}")
```

Input:

```
Enter 2 numbers separated by space:12 45
```

Output:

```
Array elements: [12.0, 45.0]
Sum: 57.0
Average: 28.5
```

Program No: 07
Program Name: Write a simple python program to explain classNotFound Exception and endOfFile(EOF) exception**.**
Objectives:

- To simulate and understand the Python equivalent of ClassNotFoundException using NameError.
- To demonstrate how EOFError occurs and is handled in Python when reading input.

Theory: In Python programming, exceptions are used to handle unexpected errors that may occur during program execution. This program demonstrates two specific types of exceptions that are commonly seen:

1. ClassNotFoundException Equivalent in Python:
    - In Java, ClassNotFoundException occurs when the program tries to use a class that has not been defined or found.
    - In Python, this is equivalent to a NameError, which happens when the program tries to access a variable, function, or class that has not been defined.
    - For example, trying to create an object from a class that doesn't exist will raise a NameError.
2. EOFException Equivalent in Python:
    - In Java, EOFException occurs when the program unexpectedly reaches the end of a file during input.
    - In Python, the equivalent is EOFError, which occurs when the input() function tries to read but the user presses the end-of-file key combination (Ctrl+D for Linux/macOS or Ctrl+Z followed by Enter for Windows).

To handle these exceptions, Python uses the try-except block. When an error occurs inside the try block, Python immediately jumps to the corresponding except block to manage the error without crashing the program.
This program helps students understand how to simulate and handle such exceptions in Python using simple examples.
Algorithm:

1. Start the program.
2. Simulate ClassNotFoundException equivalent:

    - Try to create an object of an undefined class.
    - Catch the NameError exception.
    - Display a message indicating the exception was caught.
3. Simulate EOFException equivalent:
    - Prompt the user for input.
    - If the user presses EOF (Ctrl+D/Ctrl+Z), catch the EOFError.
    - Display a message indicating the exception was caught.
4. End the program.

Python Code:

```
# Simulating ClassNotFoundException equivalent (NameError in Python)
def simulate_class_not_found():
    try:
        # Trying to reference a class that is not defined
        obj = UndefinedClass()  # This will raise NameError
    except NameError as e:
        print(f"ClassNotFoundException equivalent caught: {e}")

# Simulating EOFException equivalent (EOFError in Python)
def simulate_eof_error():
    try:
        # Prompting the user to input something
         print("Please type something (press Ctrl+D on Linux/macOS or Ctrl+Z then
Enter on Windows to simulate EOFError): ")
        data = input()
    except EOFError as e:
        print(f"EOFException equivalent caught: {e}")

# Main function
if __name__ == "__main__":
    # First simulate ClassNotFoundException equivalent
    simulate_class_not_found()
```

```
# Then simulate EOFException equivalent
simulate_eof_error()
```

Output:

```
Please type something (press Ctrl+D on Linux/macOS or Ctrl+Z then Enter on Windows to simulate EOFError):

^Z

EOFException equivalent caught:

PS F:\education\4th year\4th year 2nd semister\ICE-4204> python simulate_exceptions.py

ClassNotFoundException equivalent caught: name 'UndefinedClass' is not defined
```

Program No: 08
Program Name: Write a program in python that will read an input.txt file containing n positive integers and calculate addition, subtraction, multiplication and division in separate output.txt files.
Sample input: 5 5 9 8;
Sample output: Case-1: 10 0 25 1, Case-2: 17 1 72 1
Objectives:

- To read a sequence of n positive integers from a file named input.txt, group them into pairs, and perform basic arithmetic operations—addition, subtraction, multiplication, and integer division—on each pair
- To store the results of each operation in separate output files (addition.txt, subtraction.txt, multiplication.txt, division.txt) with clear labeling for each test case (e.g., Case-1, Case-2) to ensure organized and readable output for further analysis or review.

Theory: File handling and basic arithmetic operations are essential concepts in programming. This program combines both to demonstrate how to read data from a file, process it, and write results to multiple output files.

◆ 1. File Handling in Python

Python provides built-in functions such as open(), read(), and write() to perform file operations. In this program:

- The input.txt file is opened in read mode to fetch the list of positive integers.
- Four separate files (addition.txt, subtraction.txt, multiplication.txt, and division.txt) are opened in write mode to store the results of corresponding operations.

◆ 2. String and List Manipulation

The data read from the file is initially a string. The program:

- Removes unwanted characters (like ;),
- Splits the string by spaces to form a list of number strings,
- Converts the string elements into integers using map (int, ...).

◆ 3. Pairwise Arithmetic Operations

The integers are processed in pairs:
For each pair (a, b), the program calculates:

- Addition: a + b
- Subtraction: a - b
- Multiplication: a * b
- Integer Division: a // b (if b ≠ 0)

Each result is formatted with a case number (e.g., Case-1:) and stored in a list.

🔹 4. Writing to Output Files

After processing all pairs, the program writes:

- Addition results to addition.txt
- Subtraction results to subtraction.txt
- Multiplication results to multiplication.txt
- Division results to division.txt

Algorithm

```
1. Start
2. Read numbers from input.txt
3. Remove  and split numbers
4. For every 2 numbers:
      a. Add them
      b. Subtract them
      c. Multiply them
      d. Divide them (if second number ≠ 0)
5. Write each result as Case-x: add sub mul div to output.txt
6. End
```

Python Code:

```
# Function to read input, perform operations, and write output
def process_input_file():
    try:
        # Read input from input.txt
        with open("input.txt", "r") as file:
            content = file.read().strip()

        # Remove the trailing semicolon and split into integers
        content = content.rstrip(';')
        numbers = list(map(int, content.split()))

        output_lines = []
        case_number = 1

        # Process in pairs
        for i in range(0, len(numbers) - 1, 2):
            a = numbers[i]
            b = numbers[i + 1]

            add = a + b
            sub = a - b
            mul = a * b
            div = a // b if b != 0 else 'undefined'  # prevent division by zero

            output_line = f"Case-{case_number}: {add} {sub} {mul} {div}"
            output_lines.append(output_line)

            case_number += 1

        # Write results to output.txt
        with open("output.txt", "w") as file:
            for line in output_lines:
                file.write(line + "\n")

        print("Processing complete. Check 'output.txt' for results.")

    except FileNotFoundError:
        print("Error: 'input.txt' not found.")
    except ValueError:
        print("Error: Please ensure 'input.txt' contains only integers.")
    except Exception as e:
        print("An unexpected error occurred:", e)

# Run the function
```

process_input_file()

Input:

Input file is input.txt

Output:

Output file is output.txt

Program No: 09
Program Name: Explain the role of software engineering in Biomedical Engineering and in the field of Artificial Intelligence and Robotics.
Solutions:
Role of Software Engineering in Biomedical Engineering
Software engineering plays a crucial role in modern biomedical systems by ensuring that medical devices, healthcare systems, and diagnostic tools are efficient, accurate, reliable, and safe.
Key Roles:

1. Medical Imaging Software:
    - Develops software to process and analyze images from MRI, CT, and X-ray machines.
    - Enhances image quality, supports diagnosis, and enables 3D modeling of organs.
2. Health Monitoring Systems:
    - Creates software for wearable health devices (like smartwatches and fitness trackers) that monitor heart rate, glucose levels, etc.
    - Sends real-time data to healthcare providers for early intervention.
3. Electronic Health Records (EHR):
    - Engineers secure and user-friendly systems to manage patient data, prescriptions, and history.
    - Ensures compliance with data privacy laws like HIPAA.
4. Medical Simulations and Training:
    - Builds virtual environments for training doctors and students using realistic simulations.
5. Surgical Robots and Embedded Software:
    - Programs the embedded systems in robotic surgery tools, ensuring real-time performance and safety.

Role of Software Engineering in Artificial Intelligence and Robotics
Software engineering is the backbone of AI and robotics, providing the structure and logic needed for intelligent behavior, learning, and control.
Key Roles:

1. AI Algorithms and Machine Learning Models:

    - Engineers design and implement models for tasks like image recognition, natural language processing, and prediction systems.
2. Robot Control Systems:
    - Develops software for motion planning, sensor integration, and decision-making in autonomous robots.
    - Ensures the robot can interact with its environment safely and effectively.
3. Simulation and Testing:
    - Builds simulation tools to test AI behavior and robotic actions before real-world deployment.
4. Human-Robot Interaction (HRI):
    - Designs intuitive software interfaces so humans can communicate with robots effectively.
5. Autonomous Systems:
    - Writes software for self-driving cars, drones, and service robots using AI, computer vision, and real-time data processing.

Program No: 10
Program Name: Study the various phases of Water-fall model. Which phase is the most dominant one?
Solution:
The Waterfall Model is a linear and sequential software development methodology where the development process flows downwards like a waterfall through several distinct phases. Each phase must be completed before the next begins, and there is little or no overlap between phases.
It was one of the earliest models used in software engineering and is best suited for projects with clearly defined requirements that are not expected to change.

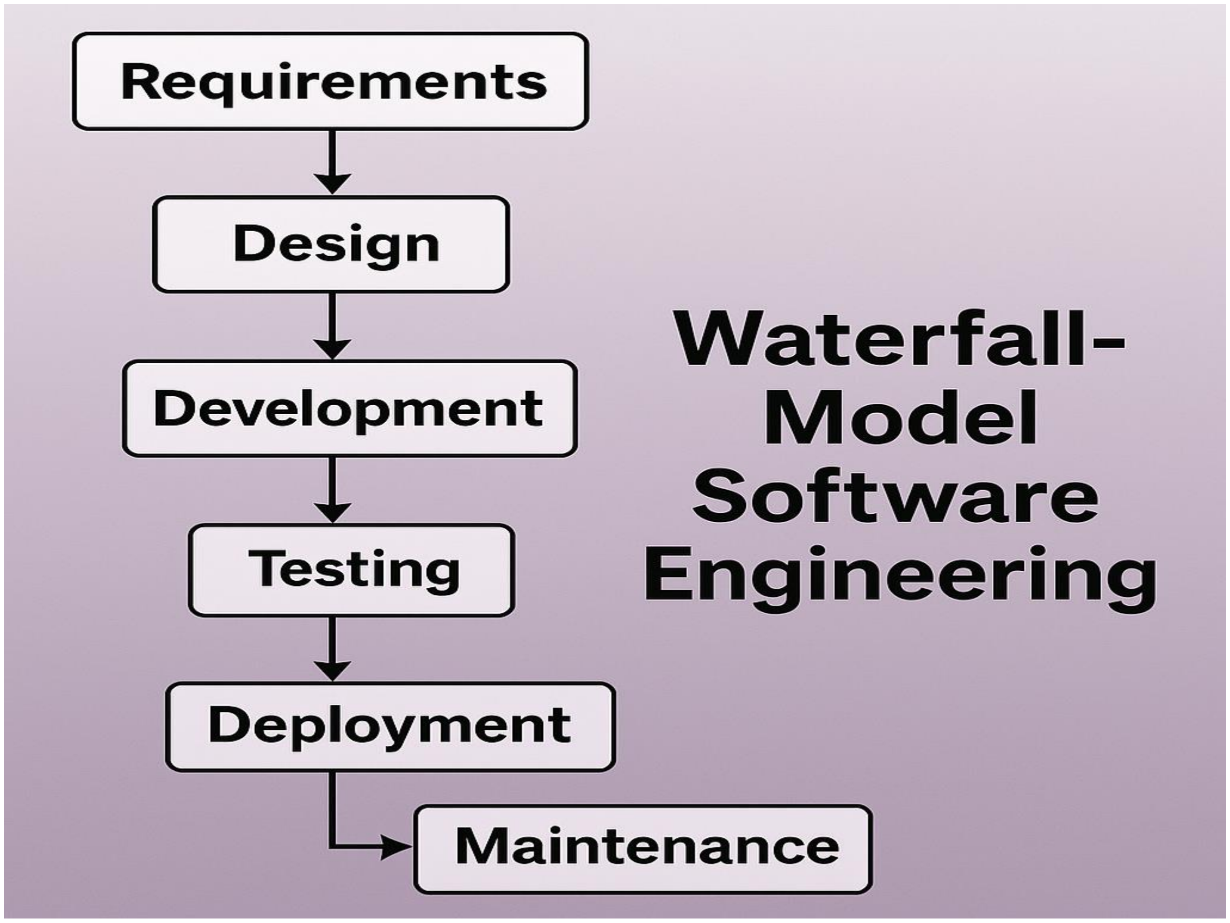

Figure 1: Block diagram of All the Phases of the Waterfall Model

Phases of the Waterfall Model

1. Requirements Analysis and Specification

- Purpose: Gather and document all software requirements from the client or end-users.
- Output: A detailed Software Requirements Specification (SRS) document.
- Note: Any mistake here will carry over to all later stages.

2. Design

- Purpose: Plan the system architecture based on the SRS.
- Two levels:
    - High-Level Design (HLD): Overall system architecture.
    - Low-Level Design (LLD): Details of each module/component.
- Output: Software Design Document (SDD).

3. Implementation (Coding)

- Purpose: Translate the design into source code.
- Activity: Developers write code for all modules based on the design specs.
- Tools: Programming languages, compilers, and IDEs.

4. Testing

- Purpose: Validate the functionality and correctness of the software.
- Types:
    - Unit Testing
    - Integration Testing
    - System Testing
    - Acceptance Testing
- Goal: Identify and fix defects before deployment.

5. Deployment

- Purpose: Release the tested software to users or clients.
- Includes: Installation, configuration, and user training.

6. Maintenance

- Purpose: Modify the software after delivery to correct faults, improve performance, or adapt to changes.
- Types:
    - Corrective: Fix bugs.
    - Perfective: Improve performance or add new features.
    - Adaptive: Update software for a new environment.
- Fact: Maintenance consumes around 60% of the total project effort, making it the most dominant phase in the Waterfall model.

In the Waterfall Model, the most dominant phase is the **Maintenance phase**. This phase begins after the software has been developed, tested, and deployed, and it continues for the longest duration—often for several years—as long as the software remains in use. The maintenance phase includes activities such as correcting errors (corrective maintenance), adapting the software to new environments (adaptive maintenance), enhancing performance or features (perfective maintenance), and making the system more robust (preventive maintenance). It is considered the most dominant phase because it typically consumes the highest amount of time, effort, and resources, often accounting for more than 60% of the total project cost. Additionally, during this phase, real user feedback leads to continuous changes and improvements, making it a critical and ongoing part of the software development lifecycle.

Program No: 11
Program Name: Using COCOMO model estimate effort for specific problem in industrial domain.
Objectives:

- To estimate the software development effort (person-months) for an industrial domain project using the COCOMO model.
- To determine the development time and resource requirements based on software size (KLOC) and project type using the Basic COCOMO model for industrial software projects.

Theory: COCOMO (Constructive Cost Model) is a widely used algorithmic software cost estimation model developed by Barry Boehm. It estimates the effort (person-months), cost, and schedule for software development based on the size of the software (usually in Kilo Lines of Code - KLOC).
COCOMO Basic Model
The Basic COCOMO estimates effort using the formula:
Effort (person-months) $= a \times (\mathrm{KLOC})^{\mathrm{b}}$
where:

- KLOC = estimated size of the software in thousands of lines of code.
- a, b are constants depending on the mode of the project.

COCOMO Modes & Parameters

Mode	Description	a	b
Organic	Small, simple software in familiar domains (e.g., small industrial projects)	2.4	1.05
Semi-detached	Medium size, mixed experience team	3.0	1.12
Embedded	Complex projects with tight constraints (industrial control systems, real-time)	3.6	1.20

Step-by-step example for Industrial Domain
Since you mention "industrial domain," it often fits Embedded mode due to complexity and constraints (e.g., real-time control systems, embedded software).

Suppose:

- Estimated software size = 50 KLOC (50,000 lines of code)
- Mode = Embedded (industrial domain)

Effort estimation:

Effort = $a \times (\mathrm{KLOC})^{\mathrm{b}} = 3.6 \times (50)^{1.20} = 393.5\, person - months$

Interpretation:

- Estimated effort is 393.5 person-months.
- This means about 393.5 months of work by one person or for example, about 33 people working for one year (assuming 12 months per year).

Optional: Development Time Estimation (Basic COCOMO)

Development Time (months) $\mathrm{T} = c \times (\mathrm{Effort})^{\mathrm{d}}$ Where c=2.5, d=0.32

$\mathrm{T} = 2.5 \times (393.5)^{.32} = 16.93$ months

Algorithm: COCOMO Effort and Time Estimation

1. Start
2. Define COCOMO model parameters (a, b, c, d) for three modes: organic, semi-detached, and embedded.
3. Input:
   - Estimated software size in KLOC (e.g., 50)
   - Select the mode (embedded for industrial domain)
4. Check if the mode is valid.
5. Retrieve the constants (a, b, c, d) for the selected mode.
6. Calculate Effort using the formula:
   $$\mathrm{Effort} = a \times (\mathrm{KLOC})^{\mathrm{b}}$$
7. Calculate Development Time using the formula:
   $$\mathrm{Time} = c \times (\mathrm{Effort})^{\mathrm{d}}$$
8. Output the estimated Effort and Development Time.
9. End

Python Code:

```
import math
def cocomo_basic(kiloc, mode):
    # Define parameters for each mode
    modes = {
        'organic':    {'a': 2.4, 'b': 1.05, 'c': 2.5, 'd': 0.38},
        'semi-detached': {'a': 3.0, 'b': 1.12, 'c': 2.5, 'd': 0.35},
        'embedded':   {'a': 3.6, 'b': 1.20, 'c': 2.5, 'd': 0.32}
    }
    if mode not in modes:
        raise ValueError(f"Mode must be one of {list(modes.keys())}")

    params = modes[mode]
    a, b, c, d = params['a'], params['b'], params['c'], params['d']

    # Effort estimation (person-months)
    effort = a * (kiloc ** b)

    # Development time (months)
    time = c * (effort ** d)

    return effort, time

# Example usage for industrial domain (embedded mode)
software_size_kloc = 50  # example size in KLOC
mode = 'embedded'
effort, development_time = cocomo_basic(software_size_kloc, mode)
print(f"COCOMO Basic Model - Mode: {mode.capitalize()}")
print(f"Software Size: {software_size_kloc} KLOC")
print(f"Estimated Effort: {effort:.2f} person-months")
print(f"Estimated Development Time: {development_time:.2f} months")
```

Output:

```
COCOMO Basic Model - Mode: Embedded
Software Size: 50 KLOC
Estimated Effort: 393.61 person-months
Estimated Development Time: 16.92 months
```

Program No: 12

Program Name: Identify the reasons behind software crisis and explain the possible solutions for the following scenario:

- Case 1: Air ticket reservation software was delivered to the customer and was installed in an airport 12.00 AM (midnight) as per the plan. The system worked quite fine till the next day 12.00 PM (noon). The system crashed at 12.00 PM and the airport authorities could not continue using software for ticket reservation till 5.00 PM. It took 5 hours to fix the defect in the software.
- Case 2: Software for financial systems was delivered to the customer. Customer conformed the development team about a mal-function in the system. As the software was huge and complex, the development team could not identify the defect in the software.

Solution:

Reasons Behind Software Crisis

Software crisis refers to the challenges in software development, including delays, budget overruns, low-quality products, and maintenance difficulties. The main reasons include:

- **Poor Software Design:** Inadequate planning and architecture lead to unscalable or error-prone systems.
- **Insufficient Testing:** Lack of thorough testing can cause bugs to appear in real-world use.
- **Inadequate Requirements Analysis:** Unclear or changing requirements cause mismatched user expectations.
- **Lack of Maintenance Planning:** Many systems are deployed without proper plans for bug fixes and updates.
- **Complexity of Large Systems:** Bigger systems are difficult to debug and manage without proper modular design.
- **Time and Cost Constraints:** Pressures to release software quickly may lead to skipping essential development phases.

Case-wise Analysis and Solutions

Case 1: Air Ticket Reservation Software Crash at 12:00 PM

- **Problem:**
  Software worked from 12:00 AM (midnight) till 12:00 PM (noon), and

crashed. It took 5 hours to fix, affecting ticketing services during peak operational hours.

- **Possible Causes:**
    - Time-related bug (e.g., mishandling of AM/PM formats).
    - Improper handling of edge cases like noon (12:00 PM).
    - Inadequate real-time system testing.
- **Solutions:**

1. **Comprehensive Time Testing:** Simulate real-time usage covering boundary values like 12:00 AM and 12:00 PM.
2. **Implement Robust Time Libraries:** Use standard time libraries to avoid errors related to formatting and parsing.
3. **Automated Monitoring Tools:** Include crash detection and alert systems to reduce downtime.
4. **Failover Mechanism:** Develop a backup or mirrored system to switch during system failure.

Case 2: Malfunction in Financial Software

- **Problem:**
  The delivered financial software had a malfunction. Due to its complexity, the development team failed to locate the defect.
- **Possible Causes:**
    - Lack of modular design making the software difficult to debug.
    - Poor documentation and logging.
    - No automated testing or error reporting.

- **Solutions:**

1. **Modular Architecture:** Design the system in smaller, testable units to isolate problems easily.
2. **Effective Logging:** Integrate detailed logs to trace errors efficiently.
3. **Regular Code Reviews:** Ensure different team members regularly review code to identify potential issues.
4. **Unit and Integration Testing:** Run automated tests on each module before release.
5. **Use of Debugging Tools:** Implement debugging tools that help locate errors quickly in large codebases.

Software crises arise primarily due to lack of planning, insufficient testing, and complexity in system design. To avoid such crises:

- Adopt standard development methodologies (like Agile or DevOps),
- Focus on robust testing,
- Maintain good documentation,
- And ensure scalable, modular software design.

These practices significantly reduce the chances of failure and simplify maintenance in real-world industrial scenarios.