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

Date: 21-05-2025 Signature………………

Submitted by
Md. Sohel Rana
Roll:200608
Session:2019-2020
4th year 2nd semester
Department of Information and Communication Engineering
Pabna University of Science and Technology

Submitted To
Md. Anwar Hossain
Professor
Department of Information and Communication Engineering
Pabna University of Science and Technology

INDEX

Pb. No.	Problem Name	Page No.
01	Write a program in "JAVA" or "C" to develop a simple calculator that would be able to 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 "=" sign. Sample input: 1+2; 8%4; Sample output: 1+2=3; 8%4=0.	
02	Write a program in "JAVA" or "C" 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.	
03	Write a program in "JAVA" or "C" to check weather 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. Program user will insert a number or string as input directly and the program will display the exact result in the output console.	
04	Write down the ATM system specifications and report the various bugs.	
05	Write a program in "JAVA" or "C" to find out the factorial of a number using while or for loop. Also verify the results obtained from each case.	
06	Write a program in "JAVA" or "C" that will find sum and average of array using do while loop and 2 user defined function.	
07	Write a simple "JAVA" program to explain classNotFound Exception and endOfFile(EOF) exception.	
08	Write a program in "JAVA" or "C" that will read a input.txt file containing n positive integers and calculate addition, subtraction, multiplication and division in separate output.txt file. Sample input: 5 5 9 8; Sample output: Case-1:10 0 25 1; Case-2: 17 1 72 1.	
09	Explain the role of software engineering in Biomedical Engineering and in the field of Artificial Intelligence and Robotics.	
10	Study the various phases of Water-fall model. Which phase is the most dominated one?	
11	Using COCOMO model estimate effort for specific problem in industrial domain	
12	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 (mid night) 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".	

Problem No. 1

Problem Name: Write a program in "JAVA" or "C" to develop a simple calculator that would be able to

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

✅ Objective:

To develop a simple calculator in C that:

- Takes two numbers, an arithmetic operator (`+`, `-`, `*`, `/`, `%`) and an equals sign (=) as input.
- Performs the specified operation.
- Displays the result in the format: `num1 operator num2 = result`.

□ Algorithm:

1. Start the program.
2. Declare variables:
    - `num1` and `num2` as `double` for general numeric input.
    - `charecter` to store the arithmetic operator.
    - `press` to store the equal sign.
3. Prompt the user to enter `num1`.
4. Read `num1` using `scanf`.
5. Prompt the user to enter `num2`.
6. Read `num2` using `scanf`.
7. Prompt the user to enter the operator (`+`, `-`, `*`, `/`, `%`).
8. Read the operator using `scanf` (with space before `%c` to consume any leftover newline).
9. Prompt the user to enter =.
10. Read the = sign.
11. Check if the input character is =.
12. If yes, then:
    - Use `if-else` conditions to check which operator was entered.
    - Perform the corresponding operation.
    - For `%`, cast both `num1` and `num2` to `int` before using the modulo operator.
    - Display the result in the format: `num1 operator num2 = result`.
    - Handle division or modulo by zero with an error message.

13. If operator is invalid, display an error.
14. End the program.

Source code

```
 #include<bits/stdc++.h>
using namespace std;
int main(){
double num1,num2;
cout<<"Enter num1: "; cin>>num1;
cout<<"Enter num2: "; cin>>num2;
char charecter,press;
cout<<"Enter +,-,*,/,%: "; cin>>charecter;
cout<<"Enter =: "; cin>>press;
if(press=='='){
if(charecter=='+'){
cout<<num1<<"+"<<num2<<" = "<<num1+num2<<endl;
}
else if(charecter=='-'){
cout<<num1<<"-"<<num2<<" = "<<num1-num2<<endl;
}
else if(charecter=='*'){
cout<<num1<<"*"<<num2<<" = "<<num1*num2<<endl;
}
else if(charecter=='/'){
cout<<num1<<"/"<<num2<<" = "<<num1/num2<<endl;
}
else{
cout<<num1<<"%"<<num2<<" = "<<int(num1)%int(num2)<<endl;
}
}
}
```

Sample Inputs and Outputs

Example 1: Addition

```
Enter num1: 10
Enter num2: 5
Enter +,-,*,/,%: +
Enter =: =
10+5 = 15
```

▯ Example 2: Subtraction

```
Enter num1: 12.5
Enter num2: 7.2
Enter +,-,*,/,%: -
Enter =: =
12.5-7.2 = 5.3
```

▯ Example 3: Multiplication

```
Enter num1: 4
Enter num2: 3
Enter +,-,*,/,%: *
Enter =: =
4*3 = 12
```

▯ Example 4: Division

```
Enter num1: 10
Enter num2: 2
Enter +,-,*,/,%: /
Enter =: =
10/2 = 5
```

▯ Example 5: Modulo

```
Enter num1: 9
Enter num2: 4
Enter +,-,*,/,%: %
Enter =: =
9%4 = 1
```

Problem No.2

Problem Name: Write a program in "JAVA" or "C" 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.

✓ Objective:

To create a C++ program that:

- Takes `n` pairs of integers (total `2n` numbers).

- Then takes an arithmetic operator (`+`, `-`, `*`, `/`) as input.
- Performs the chosen operation on each consecutive pair.
- Outputs the result of each operation.

Algorithm:

1. Start the program.
2. Ask the user for the number `n` (which determines how many pairs of numbers will be input).
3. Declare a `vector` to hold `2n` numbers (since each pair has 2 numbers).
4. Use a loop to input `2n` numbers and store them in the vector.
5. Ask the user to input a single operator (`+`, `-`, `*`, `/`).
6. Use conditional statements (`if`) to check the operator.
7. For each pair of numbers (i.e., index `i` and `i+1` in steps of 2), perform the selected operation.
8. Display the result for each pair.
9. End the program.

Source code:

```
Source Code:
#include <bits/stdc++.h>
using namespace std;
int main()
{
int n;
cout<<"Enter number: "; cin>>n;
vector<double>vec(2*n);
int j = 2*n;
cout<<"Enter numbers: "<<endl;
for(int i=0; i<j; i++)
cin>>vec[i];
char ch;
cout<<"Enter operator: "; cin>>ch;
if(ch == '+'){
for(int i=0; i<j; i+=2){
cout<<vec[i]<<" + "<<vec[i+1]<<" = "<<vec[i] + vec[i+1]<<endl;}}
if(ch == '-'){
for(int i=0; i<j; i+=2){
cout<<vec[i]<<" - "<<vec[i+1]<<" = "<<vec[i] - vec[i+1]<<endl;}}
if(ch == '*') {
for(int i=0; i<j; i+=2) {
cout<<vec[i]<<" * "<<vec[i+1]<<" = "<<vec[i] * vec[i+1]<<endl;}}
if(ch == '/') {
for(int i=0; i<j; i+=2) {
cout<<vec[i]<<" / "<<vec[i+1]<<" = "<<vec[i] / vec[i+1]<<endl;}}
return 0;
}
```

Sample Input:
Enter number: 3
Enter numbers:
4 5 7 8 20 40
Enter operator: +

✅*Sample Output:*

CopyEdit
4 + 5 = 9
7 + 8 = 15
20 + 40 = 60

Sample Input:
Enter number: 2
Enter numbers:
100 20 30 5
Enter operator: -

✅ Sample Output:

100 - 20 = 80
30 - 5 = 25

✅**Sample Input***:*
Enter number: 2
Enter numbers:
6 3 10 2
Enter operator: /

✅ Sample Output:

6 / 3 = 2 10 / 2 = 5

Problem No.3

Problem name: Write a program in "JAVA" or "C" to check weather 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. Program user will insert a number or string as input directly and the program will display the exact result in the output console.

✓ Objective

To develop a console program (in C++ or Java) that:

- Lets the user decide whether to check a string or a number for palindrome.
- Reads a single input (word or integer) directly—no test-case count.
- Determines if the entered string or number reads the same forwards and backwards.
- Prints a clear message indicating whether it is or is not a palindrome.

Algorithm

1. Start the program.
2. Prompt the user:
3. `Enter 's' for string checking or 'n' for number checking:`
4. Read the choice character `ch`.
5. If `ch == 's'` (string mode):
    1. Prompt:
    2. `Enter a word without space:`
    3. Read `string1`.
    4. Copy it to `string2`.
    5. Reverse `string2`.
    6. If `string1 == string2`, print
    7. `The given word is palindrome.`

       Else, print

       `The given word is not palindrome.`
6. Else if `ch == 'n'` (number mode):
    1. Prompt:
    2. `Enter number without space:`
    3. Read integer `number`.
    4. Convert `number` → string → `number1`.
    5. Copy it to `number2`.
    6. Reverse `number2`.
    7. If `number1 == number2`, print
    8. `The given number is palindrome.`

       Else, print

       `The given number is not palindrome.`

7. End the program.

Source code:

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
string string1, string2, number1,number2;
int number;
char ch;
cout<<"Enter 's' for string checking or 'n' for number checking: "; cin>>ch;
if(ch == 's'){
cout<<"Enter a word without space: "; cin>>string1;
string2 = string1;
reverse(string2.begin(), string2.end());
if(string1 == string2){
cout<<"The given word is palindrome."<<endl;
}
else{
cout<<"The given word is not palindrome."<<endl;
}
}
if(ch == 'n'){
cout<<"Enter number without space: "; cin>>number;
number1 = to_string(number);
number2 = number1;
reverse(number2.begin(), number2.end());
if(number1 == number2){
cout<<"The given number is palindrome."<<endl;
}
else{
cout<<"The given number is not palindrome."<<endl;
}
}
return 0;
}
```

▯ Input/Output Examples

Example 1: String Palindrome

```
Enter 's' for string checking or 'n' for number checking: s
Enter a word without space: radar
The given word is palindrome.
```

Example 2: String Non-Palindrome

```
Enter 's' for string checking or 'n' for number checking: s
Enter a word without space: hello
The given word is not palindrome.
```

Example 3: Number Palindrome

```
Enter 's' for string checking or 'n' for number checking: n
Enter number without space: 12321
The given number is palindrome.
```

Example 4: Number Non-Palindrome

```
Enter 's' for string checking or 'n' for number checking: n
Enter number without space: 12345
The given number is not palindrome.
```

Problem No.4

Problem name: Write down the ATM system specifications and report the various bugs.

Answer:

❖ **ATM System Features**

1. Login and PIN Check:
   - The user puts their ATM card into the machine.
   - The ATM asks the user to enter their 4-digit PIN.
   - The PIN is checked with the bank's records.
   - If the PIN is wrong, the user gets 3 chances. After 3 wrong tries, the card is blocked.
2. Main Menu Options (After Login):
   - Withdraw Cash
   - Deposit Cash
   - Check Balance
   - Transfer Money
   - Change PIN
   - Mini Statement
   - Exit
3. Withdrawing Cash:
   - The user chooses "Withdraw Cash" and enters the amount.
   - The ATM checks if the user has enough money in their account.
   - The ATM also checks if it has enough cash inside.
   - If all is fine, it gives out the cash and updates the account balance.
   - A receipt is printed if the user wants it.
4. Depositing Money:
   - The user chooses "Deposit Cash" and enters the amount.
   - The ATM asks the user to put the cash or checks in the deposit slot.
   - After counting, the amount is added to the account.

   - A receipt is printed if requested.
5. Checking Balance:
   - The user selects "Check Balance."
   - The ATM shows the current balance from the bank’s system.
6. Transferring Money:
   - The user selects "Transfer Funds" and enters the receiver’s account number and amount.
   - The ATM checks both accounts and sends the money if possible.
7. Changing PIN:
   - The user selects "Change PIN."
   - They enter the old PIN, then a new one.
   - If correct, the new PIN is saved.
8. Mini Statement:
   - The user selects "Mini Statement."
   - The ATM prints a list of the last few transactions.
9. Errors and Security Features:
   - If there’s a problem (like paper jam or power cut), the ATM stops the action and shows an error.
   - Security includes:
     - Blocking cards after 3 wrong PIN tries
     - Encrypting data
     - Keeping a safe record of all activities

Possible Problems or Bugs in the ATM System

1. Login Problems:
   - The card gets blocked even before 3 wrong tries.
   - The system lets someone skip the PIN check.
   - Good cards may not be read correctly.
2. Money Transaction Issues:
   - Money not given but still deducted from account.
   - Wrong amount deducted.
   - Receipt not printed or shows wrong info.
3. User Interface Issues:
   - User selects one option but goes to another.
   - ATM screen freezes or works slowly.
   - Confusing or unclear messages shown to user.
4. Deposit Issues:
   - ATM counts money or checks wrongly.
   - Money is added twice or not added at all.
5. Balance Errors:
   - Shows old or wrong account balance due to update delay.
6. Transfer Problems:
   - Money goes to the wrong account even if number was correct.

   - Transfer happens more than once by mistake.
7. PIN Change Problems:
   - PIN doesn't actually get updated.
   - ATM allows very weak PINs (like “1234”).
8. Security Issues:
   - Private info shown on screen or receipt.
   - Someone can repeat a transaction without permission.
   - Wrong data shown in transaction logs.
9. Hardware or Network Issues:
   - Blocked cards don’t get taken by the machine.
   - Internet connection drops during a transaction.
   - Printer, card reader, or cash slot not working properly.
10. User Experience Problems:

- Not easy for disabled users (e.g. no voice or large text).
- On-screen instructions are not clear or helpful.

Conclusion:

To keep ATM systems safe, fast, and easy to use, banks should regularly test, update, and maintain them. Fixing bugs and improving the user interface helps people trust and use ATMs without problems.

✓ ATM System Specifications

1. Authentication:
   - The user inserts their ATM card into the machine.
   - The ATM prompts the user to enter their Personal Identification Number (PIN).
   - The system verifies the PIN with the bank’s database.
   - The user is allowed up to 3 incorrect attempts before the card is blocked.
2. Main Menu Options:
   - Withdraw Cash
   - Deposit Cash
   - Check Balance
   - Transfer Funds
   - Change PIN
   - Mini Statement
   - Exit
3. Cash Withdrawal:
   - The user selects the withdrawal option and enters the desired amount.

   - The system checks if the user has enough balance and if the ATM has enough cash.
   - If both are available, the money is dispensed and the account is updated.
   - A receipt is printed if the user requests it.
4. Cash Deposit:
   - The user selects the deposit option and enters the amount.
   - The ATM asks the user to insert cash or checks into the deposit slot.
   - After verification, the amount is added to the account.
   - A receipt is printed if requested.
5. Balance Inquiry:
   - The user selects "Check Balance."
   - The system displays the current account balance.
6. Funds Transfer:
   - The user selects "Transfer Funds" and provides the recipient's account number and the amount.
   - The system verifies both accounts and completes the transfer if valid.
7. PIN Change:
   - The user selects "Change PIN."
   - The current PIN and the new PIN are entered.
   - If the current PIN is correct, the new one is saved.
8. Mini Statement:
   - The user selects "Mini Statement."
   - The ATM prints a list of recent transactions.
9. Error Handling & Security:
   - In case of errors, the ATM stops the process and shows an error message.
   - Features include:
     - Card blocking after failed login
     - Encrypted data handling
     - Secure logging of all transactions

❌ Various Bugs in the ATM System

1. Authentication Bugs

- The card gets blocked before 3 incorrect PIN attempts.
- Users can bypass PIN verification.
- Valid cards may not be read due to card reader failure.

2. Transaction Bugs

- Cash not dispensed, but amount deducted from the account.
- Wrong amount deducted during withdrawal or transfer.
- Receipt not printed or has wrong details.

3. Interface Bugs

- Wrong screen appears after selecting an option.
- Slow or frozen screen during use.
- Messages shown are confusing or incorrect.

4. Deposit Bugs

- ATM miscounts or does not detect inserted cash/checks.
- Deposited money is not added or is added twice.

5. Balance Inquiry Bugs

- Wrong or outdated balance shown due to delay in updates.

6. Transfer Bugs

- Money is sent to the wrong account even with correct input.
- Transfer happens more than once due to system error or repeated clicks.

7. PIN Change Bugs

- PIN change does not save properly.
- Weak PINs are allowed (e.g., too short or too simple).

8. Security Bugs

- Sensitive details shown on screen or receipt.
- Unauthorized users can repeat a previous transaction.
- Incorrect transaction logs (successful marked as failed or vice versa).

9. Hardware & Network Bugs

- ATM fails to keep a blocked or expired card.
- Network drops cause failed or incomplete transactions.
- Devices like cash dispenser or card reader stop working often.

10. User Experience Bugs

- Not usable for people with disabilities (e.g., no voice guidance, poor contrast).
- Instructions are not easy to understand, leading to user mistakes.

Problem No.5

Problem Name: Write a program in "JAVA" or "C" to find out the factorial of a number using while or for

loop. Also verify the results obtained from each case.

✓ Objective:

To write a program that calculates the **factorial of a given number** using a **`for` loop** and displays the **step-by-step multiplication process** for verification.

Algorithm:

1. Start the program.
2. Prompt the user to enter a number (`number`).
3. Initialize `fact = 1` (to store factorial) and `j = 1` (for tracking iterations).
4. Use a `for` loop starting from `i = number` down to `1`.
5. Inside the loop:
   - Print the current step multiplication.
   - Multiply `fact` by `i`.
   - Increment `j`.
6. After the loop ends, display the final result `number! = fact`.
7. End the program.

Source code:

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    int number;
    cout << "Enter a number to find factorial: ";
    cin >> number;

    long long int fact = 1;
    int j = 1;

    for (int i = number; i > 0; i--) {
        cout << "Factorial in iteration " << j << " is : "
             << fact << " * " << i << " = " << fact << endl;
        fact = fact * i;
        j += 1;
    }

    cout << endl << number << "! = " << fact << endl;
    return 0;
}
```

Input:

```
Enter a number to find factorial: 5
```

Output:

```
Factorial in iteration 1 is : 1 * 5 = 1
Factorial in iteration 2 is : 5 * 4 = 5
Factorial in iteration 3 is : 20 * 3 = 20
Factorial in iteration 4 is : 60 * 2 = 60
Factorial in iteration 5 is : 120 * 1 = 120

5! = 120
```

Problem No.6

Problem name: Write a program in "JAVA" or "C" that will find sum and average of array using do while loop and 2 user defined function.

✅ **Objective:**

To write a program in C **(or** Java**) that:**

- Accepts an array of integers from the user.
- Calculates the **sum** and **average** of the array elements using a `do-while` loop.
- Uses two **user-defined functions**: one for sum and one for average.

Algorithm:

1. Start the program.
2. Prompt the user to input the size of the array (`n`).
3. Declare an array of size `n`.
4. Use a loop to read `n` integer elements into the array.
5. Define a function `sum()`:
    - Initialize `i = 0`, `sum = 0`.
    - Use a `do-while` loop to add each element to `sum`.
    - Return the result.

6. Define a function `avg()`:
    - Call the `sum()` function to get the sum.
    - Divide the sum by number of elements to get average.
7. Display the sum and average in `main()`.
8. End the program

Souce code:

```
#include <iostream>
#include <vector>
using namespace std;

// Function to calculate sum using do-while loop
int sum(const vector<int>& arr) {
   int i = 0;
   int total = 0;
   do {
      total += arr[i];
      i++;
   } while (i < arr.size());
   return total;
}

// Function to calculate average using do-while loop
float avg(const vector<int>& arr) {
   int total = sum(arr);  // reuse sum function
   return static_cast<float>(total) / arr.size();
}

int main() {
   int n;
   cout << "Enter array size: ";
   cin >> n;

   vector<int> arr(n);
   cout << "Enter array elements: ";
   for (int i = 0; i < n; i++) {
      cin >> arr[i];
   }

   int total = sum(arr);
   float average = avg(arr);
```

```
    cout << "Sum : " << total << endl;
    cout << "Average : " << average << endl;

    return 0;
}
```

Input:

```
Enter array size: 5
Enter array elements: 10 20 30 40 50
```

📤 Output:

```
Sum : 150
Average : 30.00
```

Problem No.7

Problem name: Write a simple "JAVA" program to explain classNotFound Exception and endOfFile(EOF) exception.

Objective:

The program demonstrates the handling of two specific exceptions in Java:

1. **ClassNotFoundException** — occurs when the JVM tries to load a class that does not exist.
2. **EOFException** — occurs when the end of a file is reached unexpectedly while reading input.

It attempts to load a non-existent class and read from a non-existent file, showing how these exceptions can be caught and handled gracefully.

Algorithm:

1. **ClassNotFoundException Handling:**
   - Use `Class.forName("NonExistentClass")` to try loading a class that doesn't exist.
   - Catch `ClassNotFoundException` and display an error message.
2. **EOFException Handling:**

- Attempt to open a file called `"nonexistent.txt"` using a `DataInputStream`.
- If the file is not found, catch `FileNotFoundException` and display a message.
- If the file exists, read bytes in a loop:
    - Convert each byte to a character and print it.
    - If `EOFException` occurs (end of file reached), print a message and exit loop.
    - Catch any other `IOException` and print stack trace.
- In the `finally` block, close the input stream safely.

Source code:

```
import java.io.*;

public class lab7 {

public static void main(String[] args) {

DataInputStream inputStream=null;

// Example 1: ClassNotFoundException

try {

Class.forName("NonExistentClass");

} catch (ClassNotFoundException e) {

System.out.println("ClassNotFoundException occurred: " + e.getMessage());

}

// Example 2: EOFException

try {

inputStream = new DataInputStream(new FileInputStream("nonexistent.txt"));

while (true) {

try {

char c = (char)inputStream.readByte();
System.out.print(c);

} catch (EOFException eof) {

System.out.println("\nEnd of file reached.");

break;

} catch (IOException ioe) {

ioe.printStackTrace();

break;

}

}

} catch (FileNotFoundException fnfe) {

System.out.println("File not found: nonexistent.txt");

} finally {

try {

inputStream.close();

} catch (IOException ioe) {

ioe.printStackTrace();

} } } }
```

Input:

- No user input is required.
- The program tries to open a file named `"nonexistent.txt"` which does not exist (intentionally to show exception handling).

Output:

```
ClassNotFoundException occurred: NonExistentClass
```

```
File not found: nonexistent.txt
```

Problem No.8

Problem name: Write a program in "JAVA" or "C" that will read a input.txt file containing n positive integers and calculate addition, subtraction, multiplication and division in separate output.txt file.

Sample input: 5 5 9 8; Sample output: Case-1:10 0 25 1; Case-2: 17 1 72 1.

Objective:

Write a program that reads multiple pairs of positive integers from an input file (`input.txt`), performs arithmetic operations (addition, subtraction, multiplication, division) on each pair, and writes the results for each pair into an output file (`output.txt`).

Algorithm:

1. Open `input.txt` for reading and `output.txt` for writing.
2. Read an integer `testcase` which indicates the number of pairs to process.
3. For each test case:
   - Read two integers, `Num1` and `Num2`.
   - Perform the following operations:
     - Addition: `Num1 + Num2`
     - Subtraction: `Num1 - Num2`
     - Multiplication: `Num1 * Num2`
     - Division: `Num1 / Num2` (integer division)
4. Write the result of each operation for every test case to `output.txt` in a formatted manner.
5. Close the files and end the program.

Source code:

```
#include <bits/stdc++.h>

using namespace std;

int main() {

freopen("input.txt","r",stdin);
freopen("output.txt","w",stdout);

int testcase; cin>>testcase;

vector<int> v;

while(testcase--)

{

int Num1,Num2;

cin>>Num1>>Num2;

v.push_back(Num1);

v.push_back(Num2);

}

cout<<"Sum of Two number : "<<v[0]+v[1]<<endl;

cout<<"Subtraction of Two number: "<<v[2]-v[3]<<endl;

cout<<"Multiplication of Two number: "<<v[4]*v[5]<<endl;

cout<<"Division of Two number : "<<v[6] / v[7]<<endl;

return 0;

}
```

Input (input.txt):

```
2
5  5
9  8
```

Output (output.txt):

```
Case-1: 10 0 25 1
Case-2: 17 1 72 1
```

- Addition = 17, Subtraction = 1, Multiplication = 72, Division = 1.

Problem No.9

Problem name: Explain the role of software engineering in Biomedical Engineering and in the field of Artificial Intelligence and Robotics.

Answer:

Role of Software Engineering in Biomedical Engineering

Software engineering plays a vital role in biomedical engineering by providing the essential tools and methodologies required to design, develop, and maintain software solutions for healthcare and medical applications. The integration of software engineering accelerates advancements in patient care, diagnosis, treatment, and medical research. Key roles include:

1. **Medical Device Software Development:**
   Biomedical engineers design and develop medical devices such as MRI machines, CT scanners, and pacemakers. Software engineering is critical for developing the embedded systems and control software that ensure these devices operate safely, effectively, and reliably.
2. **Clinical Decision Support Systems (CDSS):**
   Software engineering enables the creation of CDSS, which assist healthcare providers in making data-driven decisions. These systems use algorithms, machine learning, and data analytics to provide insights, predict patient outcomes, and recommend treatment plans.
3. **Health Informatics:**
   The development of Electronic Health Records (EHR) systems, telemedicine applications, and health management software depends on software engineering. These systems manage patient data, facilitate remote consultations, and streamline communication among healthcare professionals.
4. **Medical Imaging and Signal Processing:**
   Biomedical engineers work on software that processes and analyzes medical images (e.g., MRI, CT scans) and biomedical signals (e.g., ECG, EEG). Advanced techniques, including machine learning and computer vision, are used to enhance images, segment regions, and detect diseases.
5. **Wearable Health Technology and Mobile Health Apps:**
   The creation of wearable devices (e.g., fitness trackers, glucose monitors) and mobile health applications heavily relies on software engineering for building user interfaces, data collection mechanisms, and real-time monitoring features.
6. **Simulations and Modeling:**
   Software engineering supports the development of simulations and models of biological

systems, which are crucial for understanding complex physiological processes and conducting virtual experiments.

Role of Software Engineering in Artificial Intelligence and Robotics

Software engineering forms the foundation of Artificial Intelligence (AI) and Robotics by enabling the development of software that can learn, adapt, and perform tasks autonomously. Its key roles include:

1. **Development of AI Algorithms and Models:**
   Software engineering provides the frameworks and tools to design, develop, and deploy AI algorithms such as machine learning models, deep learning networks, natural language processing, and computer vision systems. Ensuring efficiency, scalability, and security of these models is essential.
2. **Robotics Control Software:**
   Robotics depends on software to control robot behavior, including motion planning, manipulation, navigation, and perception. Software engineering guarantees that these control systems are robust, real-time, and adaptable to different environments.
3. **Integration of Sensors and Actuators:**
   Software engineering is crucial in integrating sensors (e.g., LIDAR, cameras, gyroscopes) and actuators (e.g., motors, grippers) in robots. This involves writing low-level drivers, data processing algorithms, and middleware for seamless hardware communication.
4. **AI-Driven Robotics:**
   Developing AI-driven robotic systems capable of perceiving their environment, making decisions, and acting autonomously relies heavily on software engineering. Techniques such as reinforcement learning, computer vision, and natural language understanding enable robots to interact intelligently with humans and surroundings.
5. **Simulation and Testing Environments:**
   Before deployment, AI and robotic systems require extensive simulation and testing. Software engineering facilitates creating realistic simulators and virtual environments for training and testing these systems.
6. **Ethics, Security, and Compliance:**
   AI and robotics must comply with ethical standards, privacy regulations, and security protocols. Software engineers are responsible for implementing data privacy safeguards, cybersecurity measures, and ensuring legal compliance.
7. **Human-Robot Interaction (HRI):**
   Designing safe and intuitive interfaces for human-robot interaction is a critical area where software engineering contributes by developing user-friendly interfaces, real-time feedback systems, and adaptive interactive software.

Conclusion

In both Biomedical Engineering and AI/Robotics, software engineering is indispensable for developing robust, efficient, and secure systems tailored to the complex demands of these fields. It lays the groundwork for innovation, enabling advanced healthcare solutions and intelligent robotic systems that have the potential to revolutionize technology interaction and significantly improve human life.

Problem No.10

Problem name: Study the various phases of Water-fall model. Which phase is the most dominated one?

Answer:

hases of the Waterfall Model

The Waterfall Model is a traditional **software development process** that follows a linear and sequential approach. Each phase must be completed before moving to the next one.

1. Requirement Analysis

- Gather and document all the software requirements from the client or stakeholders.
- Understand what the system needs to do.
- Produce a detailed Requirement Specification Document.

2. System Design

- Use the requirements to create the system architecture and design.
- Define system hardware and software architecture, data flow, modules, and interfaces.
- Produce design documents including diagrams and technical specifications.

3. Implementation (Coding)

- Actual coding or programming is done in this phase.
- Developers translate the design into source code using a suitable programming language.
- The software is built module by module.

4. Integration and Testing

- Combine all modules and test the complete system.
- Detect and fix defects or bugs.
- Ensure the software meets the specified requirements.

5. Deployment

- The software is installed or delivered to the user.

- The system goes live in the user environment.

6. Maintenance

- Post-deployment support.
- Fix issues found after deployment.
- Make enhancements or updates as required.

Which Phase is Most Dominant?

Implementation (Coding) phase is generally the most **dominated phase** because:

- It usually takes the most time and resources.
- It directly transforms design documents into working software.
- The success of the project depends heavily on efficient coding practices.

However, some argue that **Requirement Analysis** is the most critical because errors here can propagate downstream, causing more problems later.

Problem No.11

Problem name: Using COCOMO model estimate effort for specific problem in industrial domain

Overview of the COCOMO Model

The **COCOMO (Constructive Cost Model)** is a widely-used software cost estimation model designed to estimate the **effort**, **time**, and **cost** required for software development projects. It is especially valuable in industrial domains where projects have well-defined requirements and constraints.

Purpose

COCOMO helps project managers and developers predict the resources needed to complete a software project by analyzing the project size and complexity.

Levels of the COCOMO Model

COCOMO is structured into three progressively detailed levels:

1. **Basic COCOMO**
   Provides a rough estimate of development effort based solely on the project size (measured in thousands of delivered source instructions or KLOC).

2. **Intermediate COCOMO**
   Improves estimation accuracy by incorporating various **cost drivers** that affect productivity, such as system reliability, complexity, and team experience.
3. **Detailed COCOMO (COCOMO II)**
   Builds on the intermediate model by including detailed project attributes, development environment factors, and software reuse estimates.

Basic COCOMO Estimation

The Basic COCOMO model categorizes software projects into three types, each with its own effort estimation formula:

- **Organic Projects:**
  Small teams working on relatively simple and well-understood systems with flexible requirements.
- **Semi-Detached Projects:**
  Medium-sized teams handling moderately complex systems with some experience and moderate constraints.
- **Embedded Projects:**
  Large teams working on highly complex, constrained systems with strict reliability requirements.

Effort Calculation

Effort is calculated in **person-months** using the formula based on the project type and size (KLOC).

Example: Estimating Effort for a Semi-Detached Project

Assume a software project with an estimated size of **50 KLOC** in an industrial environment, classified as a **Semi-Detached project**.

Using the Basic COCOMO formula for Semi-Detached projects, the estimated effort is approximately:

239.87 person-months

This means it would take about **240 person-months** of work to complete the project under the basic assumptions of the COCOMO model.

Conclusion

While the Basic COCOMO model provides a quick estimation, more accurate and realistic predictions require the **Intermediate** or **Detailed COCOMO** models. These models consider additional factors like team experience, project complexity, and development environment, leading to better resource planning and project management.

Problem No.12

Problem 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 (mid night) 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".

Reasons Behind the Software Crisis

The term **"software crisis"** refers to the challenges faced in software development, particularly from the 1960s to the 1980s, when many software projects were frequently **delayed, over budget, and failed to meet requirements**. The software crisis is mainly characterized by the following factors:

1. **Complexity:** Software systems have significantly increased in size and complexity, making design, development, and maintenance more difficult.
2. **Lack of Proper Requirements Analysis:** Many projects suffer from poor or incomplete requirements gathering, resulting in software that does not fulfill user needs.
3. **Poor Project Management:** Inadequate planning, insufficient documentation, and resource mismanagement contribute to project failures.
4. **Inadequate Testing:** Insufficient or ineffective testing often leads to software being released with critical bugs and defects.
5. **Rapidly Changing Requirements:** Frequent changes during development make it difficult for teams to keep pace and maintain consistency.

6. **Lack of Standardization:** Absence of industry-wide standards results in inconsistent software quality and functionality.
7. **Communication Gaps:** Miscommunication between developers, customers, and users often causes misunderstandings about software requirements.

Case Studies and Analysis

Case 1: Air Ticket Reservation Software

Scenario:

The air ticket reservation software functioned correctly until 12:00 PM the next day but then crashed, remaining unavailable for five hours.

Possible Causes of Failure:

1. **Time-Related Bug:** The defect could be triggered by a boundary condition or overflow error specific to 12:00 PM.
2. **Inadequate Testing:** The software may not have been tested for all time-bound scenarios.
3. **Resource Leakage:** Memory leaks or resource management issues might have caused the system crash.
4. **Poor Error Handling:** The system may lack mechanisms to handle edge cases, leading to a crash.
5. **Database Issues:** Problems like corrupted data or lost database connections might have occurred at the specific time.

Suggested Solutions:

1. **Thorough Testing:** Conduct comprehensive boundary, performance, and stress testing to catch time-sensitive defects.
2. **Code Review and Debugging:** Perform detailed reviews and debugging to locate and fix time-related bugs.
3. **Monitoring and Logging:** Implement logging and monitoring to trace system behavior leading to the crash.
4. **Improve Error Handling:** Enhance the software’s robustness by managing unexpected situations gracefully.
5. **Redundancy and Failover:** Employ backup systems and failover strategies to maintain service continuity.

Case 2: Software for Financial Systems

Scenario:

The software delivered to the client malfunctions, and due to its complexity, the development team struggles to identify the defect.

Possible Causes of Failure:

1. **Lack of Modularization:** A monolithic design makes it difficult to isolate and locate issues.
2. **Insufficient Documentation:** Poor documentation hampers understanding of system architecture and flow.
3. **Poor Maintenance Practices:** Lack of maintainability complicates bug fixes and updates.
4. **Complex Dependencies:** Intricate interdependencies between components obscure defect origins.
5. **Insufficient Testing:** Testing may not cover all critical edge cases, especially in complex financial systems.

Suggested Solutions:

1. **Modularize the Software:** Refactor the system into smaller modules to isolate and manage issues efficiently.
2. **Improve Documentation:** Maintain thorough documentation covering architecture, data flow, and components.
3. **Automated Testing and CI:** Use automated unit, integration, and regression tests to detect defects early.
4. **Advanced Debugging Tools:** Employ static and dynamic analysis tools and extensive logging to locate problems.
5. **Training and Knowledge Sharing:** Equip the team with best practices for debugging and modular design.
6. **Customer Collaboration:** Work closely with customers to reproduce, understand, and diagnose issues effectively.

Conclusion

By recognizing the root causes of the software crisis and implementing the recommended solutions, software projects—whether in time-sensitive systems like air ticket reservation or complex domains such as financial software—can be managed more effectively, reducing failures and improving quality and reliability.