K M M INSTITUTE OF POSTGRADUATE STUDIES

(Affiliated to Sri Venkateswara University, Tirupati)
RAMIREDDIPALLE, TIRUPATI-517 102

DEAPARTMENT OF COMPUTER APPLICATIONS

CERTIFICATE

REGISTERED NO: _____

This is to certify that _____ is a bonafide student of MCA (I Semester) course in our Institution. The student has done the Lab work as per this record for this practical fulfillment of the requirement of the MCA course ***MCA109P Computer Organization & Mathematics for Computer Application*** *(**Subject Code & Subject Title**) during the **I** Semester of the Academic Year 2025-2027.*

Head-of the Department *Lecturer-In-Charge*

Attended and submitted for the University Practical Examination.

Signature of Examiners: *1.*_____
(External Examiner-1)

*2.*_____
(External Examiner-2)

CONTENTS

1) Write a Python Menu Driven Program for Calculating Permutations & Combinations for the Given Inputs.

Aim: ToWrite a Python Menu Driven Program for Calculating Permutations & Combinations for the Given Inputs.

Description: Based on the user's choice, the program will ask for two inputs: **nnn** (total items) and **rrr** (items chosen).It will then calculate and display the result using the formulas

- **Permutation Formula**:

  P(n,r)=n!(n−r)!P(n, r) = \frac{n!}{(n - r)!}P(n,r)=(n−r)!n!

- **Combination Formula**:

  C(n,r)=n!r!(n−r)!C(n, r) = \frac{n!}{r!(n - r)!}C(n,r)=r!(n−r)!n!

- The program will allow the user to perform multiple calculations until they choose to exit.

Algorithm:

1. **Display the Menu**:
   - Show a menu with the options:
     - 1. Permutation
     - 2. Combination
     - 3. Exit
2. **Get User Choice**:
   - Read the user's input to choose between permutation, combination, or exit.
3. **If User Chooses Permutation**:
   - Prompt the user to enter values for nnn and rrr.
   - Calculate the permutation using the formula P(n,r)=n!(n−r)!P(n, r) = \frac{n!}{(n - r)!}P(n,r)=(n−r)!n!.
   - Display the result.
4. **If User Chooses Combination**:
   - Prompt the user to enter values for nnn and rrr.
   - Calculate the combination using the formula C(n,r)=n!r!(n−r)!C(n, r) = \frac{n!}{r!(n - r)!}C(n,r)=r!(n−r)!n!.
   - Display the result.
5. **If User Chooses Exit**:
   - Exit the program.
6. **Repeat**:
   - Keep repeating the process until the user chooses to exit.

Source Code:

```
import math
def permutation(n, r):
    returnmath.factorial(n) // math.factorial(n - r)
    def combination(n, r):
            returnmath.factorial(n) // (math.factorial(r) * math.factorial(n - r))
    while True:
            print("\n----- Permutation & Combination Calculator -----")
            print("1. Calculate Permutation (nPr)")
            print("2. Calculate Combination (nCr)")
    print("3. Exit")
    choice = int(input("Enter your choice (1-3): "))
    if choice == 1:
             n = int(input("Enter value of n: "))
             r = int(input("Enter value of r: "))
if r > n:
            print("Invalid Input! r cannot be greater than n.")
    else:
            print(f"\nPermutation (nPr) = {permutation(n, r)}")
    elif choice == 2:
            n = int(input("Enter value of n: "))
            r = int(input("Enter value of r: "))
if r > n:
            print("Invalid Input! r cannot be greater than n.")
    else:
            print(f"\nCombination (nCr) = {combination(n, r)}")
    elif choice == 3:
            print("Exiting the program... Thank you!")
    break
    else:
            print("Invalid choice! Please select from 1 to 3.")
```

Output:

```
----- Permutation & Combination Calculator -----
1. Calculate Permutation (nPr)
2. Calculate Combination (nCr)
3. Exit
Enter your choice (1-3): 1
Enter value of n: 7
Enter value of r: 3
Permutation (nPr) = 210
----- Permutation & Combination Calculator -----
1. Calculate Permutation (nPr)
2. Calculate Combination (nCr)
3. Exit
Enter your choice (1-3): 2
Enter value of n: 4
Enter value of r: 2
\Combination (nCr) = 6
----- Permutation & Combination Calculator -----
1. Calculate Permutation (nPr)
2. Calculate Combination (nCr)
3. Exit
Enter your choice (1-3): 3
Exiting the program... Thank you!
```

Conclusion:

The above program executed successfully

2) Write a Python Menu Driven Program to Display Truth Table for Selected Logical Connectives.

Aim: To Write a Python Menu Driven Program to Display Truth Table for Selected Logical Connectives

Description:

- The program will display a menu with the options: AND, OR, NOT, XOR, and Exit.
- The user can select a logical connective, and the program will display the truth table for the chosen operation.
- The truth table will be displayed using the two boolean values: **True** and **False** (or 1 and 0).
- The program will keep running until the user selects to exit.

Logical Connectives:

1. **AND (∧)**:
   - Truth table:
     - **True ∧ True = True**
     - **True ∧ False = False**
     - **False ∧ True = False**
     - **False ∧ False = False**
2. **OR (∨)**:
   - Truth table:
     - **True ∨ True = True**
     - **True ∨ False = True**
     - **False ∨ True = True**
     - **False ∨ False = False**
3. **NOT (¬)**:
   - Truth table:
     - **¬True = False**
     - **¬False = True**
4. **XOR (exclusive OR)**:
   - Truth table:
     - **True XOR True = False**
     - **True XOR False = True**
     - **False XOR True = True**
     - **False XOR False = False**

Algorithm:

1. **Display the Menu**: Show the available choices (AND, OR, NOT, XOR, Exit).
2. **Get User Input**: Read the user's choice.
3. **Perform Operation**:
   - For AND/OR/XOR, display the truth table for the combination of **True** and **False** values.
   - For NOT, only one value (True and False) will be used, as it negates the value.
4. **Repeat Until Exit**: The user can keep selecting different connectives or choose to exit the program.

Source Code:

```
deftruth_table_and():
print("\nA\tB\tA AND B")
for A in [0, 1]:
for B in [0, 1]:
print(f"{A}\t{B}\t{A and B}")
deftruth_table_or():
print("\nA\tB\tA OR B")
for A in [0, 1]:
for B in [0, 1]:
print(f"{A}\t{B}\t{A or B}")
deftruth_table_not():
print("\nA\tNOT A")
for A in [0, 1]:
print(f"{A}\t{int(not A)}")
deftruth_table_xor():
print("\nA\tB\tA XOR B")
for A in [0, 1]:
for B in [0, 1]:
print(f"{A}\t{B}\t{A ^ B}")
deftruth_table_nand():
print("\nA\tB\tA NAND B")
for A in [0, 1]:
for B in [0, 1]:
print(f"{A}\t{B}\t{int(not (A and B))}")
deftruth_table_nor():
print("\nA\tB\tA NOR B")
for A in [0, 1]:
for B in [0, 1]:
print(f"{A}\t{B}\t{int(not (A or B))}")
while True:
print("\n----- Logical Connectives Truth Table -----")
print("1. AND")
print("2. OR")
print("3. NOT")
print("4. XOR")
print("5. NAND")
print("6. NOR")
print("7. Exit")
choice = int(input("Enter your choice (1-7): "))
if choice == 1:
truth_table_and()
elif choice == 2:
truth_table_or()
elif choice == 3:
truth_table_not()
elif choice == 4:
truth_table_xor()
elif choice == 5:
```

```
truth_table_nand()
elif choice == 6:
truth_table_nor()
elif choice == 7:
print("Exiting Program... Thank you!")
break
else:
print("Invalid Choice! Select between 1-7.")
```

Output:

```
----- Logical Connectives Truth Table -----
1. AND
2. OR
3. NOT
4. XOR
5. NAND
6. NOR
7. Exit
Enter your choice (1-7): 1
A       B       A AND B
0       0       0
0       1       0
1       0       0
1       1       1
----- Logical Connectives Truth Table -----
1. AND
2. OR
3. NOT
4. XOR
5. NAND
6. NOR
7. Exit
Enter your choice (1-7): 2

A       B       A OR B
0       0       0
0       1       1
1       0       1
1       1       1
----- Logical Connectives Truth Table -----
1. AND
2. OR
3. NOT
4. XOR
5. NAND
6. NOR
7. Exit
Enter your choice (1-7): 3
A       NOT A
0       1
1       0
----- Logical Connectives Truth Table -----
1. AND
2. OR
3. NOT
4. XOR
5. NAND
6. NOR
7. Exit
```

```
Enter your choice (1-7): 4
A       B       A XOR B
0       0       0
0       1       1
1       0       1
1       1       0
----- Logical Connectives Truth Table -----
1. AND
2. OR
3. NOT
4. XOR
5. NAND
6. NOR
7. Exit
Enter your choice (1-7): 5
A       B       A NAND B
0       0       1
0       1       1
1       0       1
1       1       0
----- Logical Connectives Truth Table -----
1. AND
2. OR
3. NOT
4. XOR
5. NAND
6. NOR
7. Exit
Enter your choice (1-7): 6
A       B       A NOR B
0       0       1
0       1       0
1       0       0
1       1       0
----- Logical Connectives Truth Table -----
1. AND
2. OR
3. NOT
4. XOR
5. NAND
6. NOR
7. Exit
Enter your choice (1-7): 7
Exiting Program... Thank you!
```

Conclusion:

The above program executed successfully

3) Write a Python Menu Driven Program to print Reverse of Numbers for the Given Input using Recursion.

Aim: ToWrite a Python Menu Driven Program to print Reverse of Numbers for the Given Input using Recursion

Description:

- The program presents a menu with two options: reverse a number or exit.
- The user enters a number, and the program reverses the number using recursion.
- Recursion will be used to reverse the number by dividing the number step by step and reassembling it in reverse order.
- The program will continue prompting for numbers until the user chooses to exit.

Algorithm:

1. **Display the Menu**: Show options for reversing the number or exiting.
2. **Get User Input**: Read the user's choice.
3. **Reverse the Number**:
    - Use recursion to reverse the digits of the number.
    - Convert the number to a string for easy manipulation, or use mathematical operations to extract digits.
4. **Repeat Until Exit**: The program keeps running until the user selects to exit.

Recursion Logic:

To reverse a number, consider breaking it into smaller parts:

1. Extract the last digit of the number (using modulus `num % 10`).
2. Use integer division to remove the last digit (`num // 10`).
3. Reassemble the reversed number by adding the last digit to the reversed number built so far.

Source Code:

```
defreverse_number(n, rev=0):
if n == 0:
return rev
else:
returnreverse_number(n // 10, rev * 10 + n % 10)
while True:
print("\n----- Reverse Number using Recursion -----")
print("1. Reverse a Number")
print("2. Exit")
choice = int(input("Enter your choice (1-2): "))
if choice == 1:
num = int(input("Enter a number: "))
result = reverse_number(num)
print(f"Reversed Number: {result}")
elif choice == 2:
print("Exiting the program... Thank you!")
break

else:
print("Invalid choice! Please enter 1 or 2.")
```

Output:

```
----- Reverse Number using Recursion -----
1. Reverse a Number
2. Exit
Enter your choice (1-2): 1
Enter a number: 56
Reversed Number: 65
----- Reverse Number using Recursion -----
1. Reverse a Number
2. Exit
Enter your choice (1-2): 2
Exiting the program... Thank you!
```

Conclusion:

The above program executed successfully

4) Write a Python Menu Driven Program to find HCF of Given Integers using Recursions.

Aim: To Write a Python Menu Driven Program to find HCF of Given Integers using Recursions.

Description:

- The user can choose to calculate the **HCF** of two numbers or exit the program.
- The program uses recursion to find the HCF (or GCD) of two numbers using the **Euclidean algorithm**.
- The **Euclidean algorithm** for finding the HCF is based on the principle that the HCF of two numbers `a` and`b` is the same as the HCF of `b` and `a % b` (where `%` is the modulus operator).
    - The recursion continues until `b` becomes zero, at which point the HCF is `a`.
- The program will allow the user to perform multiple HCF calculations until they choose to exit.

Algorithm:

1. **Display Menu**: Show the user the choices (HCF or exit).
2. **Get User Input**: Ask the user to input two integers.
3. **Calculate HCF using Recursion**:
    - Use the recursive function to compute the HCF.
    - Apply the Euclidean algorithm: `hcf(a, b) = hcf(b, a % b)` until `b` becomes `0`.
4. **Repeat Until Exit**: The user can continue to find HCF of new pairs of numbers until they choose to exit.

Source Code

```
defhcf(a, b):
if b == 0:
return a
else:
returnhcf(b, a % b)
while True:
print("\n----- HCF Calculator using Recursion -----")
print("1. Find HCF of two numbers")
print("2. Exit")

choice = int(input("Enter your choice (1-2): "))
if choice == 1:
      num1 = int(input("Enter first number: "))
      num2 = int(input("Enter second number: "))
result = hcf(num1, num2)
print(f"HCF of {num1} and {num2} is: {result}")

elif choice == 2:
print("Exiting the program... Thank you!")
break

else:
print("Invalid choice! Please enter 1 or 2.")
```

Output:

```
----- Reverse Number using Recursion -----
1. Reverse a Number
2. Exit
Enter your choice (1-2): 1
Enter a number: 56
Reversed Number: 65
----- Reverse Number using Recursion -----
1. Reverse a Number
2. Exit
Enter your choice (1-2): 2
Exiting the program... Thank you!
```

Conclusion:

The above program executed successfully

5) Write a Python Menu Driven Program to Implement Inclusion & Exclusion Principle for the Given Input Values.

Aim:

Description:

The **Inclusion-Exclusion Principle** is a principle used in set theory to find the size of the union of two or more sets. For two sets AAA and BBB, the principle is expressed as:

$|A \cup B| = |A| + |B| - |A \cap B|$

For three sets AAA, BBB, and CCC, it is expressed as:

$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|$

The program allows the user to:

- Enter the sizes of the sets.
- Enter the sizes of the intersections (for two or three sets).
- Calculate the union size using the Inclusion-Exclusion Principle.
- Repeat the process until the user decides to exit.

Algorithm:

1. **Display the Menu**: Show options for calculating the union of two or three sets using the Inclusion-Exclusion Principle.
2. **Get User Input**: Ask for the sizes of the sets and their intersections.
3. **Calculate the Union**:
   - For two sets: $|A \cup B| = |A| + |B| - |A \cap B|$.
   - For three sets: $|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|$.
4. **Repeat Until Exit**: The user can continue to perform calculations or exit the program.

Source Code:

```
defunion_two_sets(a, b, ab):
   # Inclusion-Exclusion for two sets
return a + b - ab
defunion_three_sets(a, b, c, ab, bc, ca, abc):
   # Inclusion-Exclusion for three sets
return a + b + c - ab - bc - ca + abc
while True:
print("\n----- Inclusion & Exclusion Principle Menu -----")
print("1. Apply Inclusion-Exclusion on Two Sets")
print("2. Apply Inclusion-Exclusion on Three Sets")
print("3. Exit")
choice = int(input("Enter your choice (1-3): "))
if choice == 1:
      A = int(input("Enter |A|: "))
      B = int(input("Enter |B|: "))
      AB = int(input("Enter |A ∩ B|: "))
result = union_two_sets(A, B, AB)
print(f"|A∪ B| = {result}")
elif choice == 2:
      A = int(input("Enter |A|: "))
      B = int(input("Enter |B|: "))
      C = int(input("Enter |C|: "))
      AB = int(input("Enter |A ∩ B|: "))
      BC = int(input("Enter |B ∩ C|: "))
      CA = int(input("Enter |C ∩ A|: "))
      ABC = int(input("Enter |A ∩ B ∩ C|: "))
result = union_three_sets(A, B, C, AB, BC, CA, ABC)
print(f"|A∪ B ∪ C| = {result}")
elif choice == 3:
print("Exiting program... Thank you!")
break
else:
print("Invalid choice! Please select between 1–3.")
```

Output:

```
----- Inclusion & Exclusion Principle Menu -----
1. Apply Inclusion-Exclusion on Two Sets
2. Apply Inclusion-Exclusion on Three Sets
3. Exit
Enter your choice (1-3): 1
Enter |A|: 9
Enter |B|: 6
Enter |A ∩ B|: 3
|A ∪ B| = 12
----- Inclusion & Exclusion Principle Menu -----
1. Apply Inclusion-Exclusion on Two Sets
2. Apply Inclusion-Exclusion on Three Sets
3. Exit
Enter your choice (1-3): 2
Enter |A|: 9
Enter |B|: 7
Enter |C|: 4
Enter |A ∩ B|: 6
Enter |B ∩ C|: 5
Enter |C ∩ A|: 2
Enter |A ∩ B ∩ C|: 10
|A ∪ B ∪ C| = 17
----- Inclusion & Exclusion Principle Menu -----
1. Apply Inclusion-Exclusion on Two Sets
2. Apply Inclusion-Exclusion on Three Sets
3. Exit
Enter your choice (1-3): 3
Exiting program... Thank you!
```

Conclusion:

The above program executed successfully

6) Write a Python Menu Driven Program to Verify Two Graphs are Isomorphic or Not for the Given Vertices.

Aim: To Write a Python Menu Driven Program to Verify Two Graphs are Isomorphic or Not for the Given Vertices

Description:

•**Isomorphic Graphs**: Two graphs are said to be isomorphic if one can be transformed into the other by renaming the vertices. This means that their structures (i.e., how the vertices are connected) are the same, even if their representations (e.g., names or labels of vertices) differ.

•**Adjacency List Representation**: The graph is represented as an adjacency list, where each vertex has a list of adjacent vertices (i.e., edges connected to it).

Algorithm:

1. **Input**: The user will provide the number of vertices and edges for two graphs. Then, the adjacency lists will be given.
2. **Check for Isomorphism**: Compare the adjacency lists of both graphs.
3. **Menu-Driven Program**: The program will allow the user to:
    - Input two graphs and check if they are isomorphic.
    - Exit the program.

Source Code:

```
importitertools
defis_isomorphic(adj1, adj2, n):
vertices = list(range(n))
for perm in itertools.permutations(vertices):
match = True
for i in range(n):
for j in range(n):
if adj1[i][j] != adj2[perm[i]][perm[j]]:
match = False
break
if not match:
break
if match:
return True
return False
while True:
print("\n----- Graph Isomorphism Checker -----")
print("1. Check if two graphs are Isomorphic")
print("2. Exit")
choice = int(input("Enter your choice (1-2): "))
if choice == 1:
      n = int(input("Enter number of vertices: "))

print("\nEnter adjacency matrix for Graph 1:")
      adj1 = [list(map(int, input().split())) for _ in range(n)]
print("\nEnter adjacency matrix for Graph 2:")
      adj2 = [list(map(int, input().split())) for _ in range(n)]
ifis_isomorphic(adj1, adj2, n):
print("\nResult: Graphs are ISOMORPHIC ")
else:
print("\nResult: Graphs are NOT Isomorphic ")
elif choice == 2:
print("Exiting the program... Thank you!")
break
else:
print("Invalid choice! Please enter 1 or 2.")
```

Output:

```
----- Graph Isomorphism Checker -----
1. Check if two graphs are Isomorphic
2. Exit
Enter your choice (1-2): 1
Enter number of vertices: 3
Enter adjacency matrix for Graph 1:
0 1 1
1 0 1
1 1 0

Enter adjacency matrix for Graph 2:
0 1 1
1 0 1
1 1 0
Result: Graphs are ISOMORPHIC
----- Graph Isomorphism Checker -----
1. Check if two graphs are Isomorphic
2. Exit
Enter your choice (1-2): 1
Enter number of vertices: 4

Enter adjacency matrix for Graph 1:
0 1 1 1
1 0 1 1
0 0 1 0
0 1 0 0

Enter adjacency matrix for Graph 2:
0 1 0 0
0 0 1 0
1 0 0 0
0 0 0 1

Result: Graphs are NOT Isomorphic
----- Graph Isomorphism Checker -----
1. Check if two graphs are Isomorphic
2. Exit
Enter your choice (1-2): 2
Exiting the program... Thank you!
```

Conclusion:

The above program executed successfully

7) Write a Python Menu Driven Program to Verify whether the Given Graph is a Hamiltonian Circuit or Not From the given Adjacency Matrix.

Aim: To Write a Python Menu Driven Program to Verify whether the Given Graph is a Hamiltonian Circuit or Not From the given Adjacency Matrix

Description:

Hamiltonian Circuit: A Hamiltonian Circuit (also known as a Hamiltonian Cycle) is a cycle that visits each vertex in a graph exactly once and returns to the starting vertex.

- **Adjacency Matrix**: The graph is represented by an adjacency matrix, where `matrix[i][j] = 1` indicates an edge between vertex `i` and vertex `j`, and `matrix[i][j] = 0` indicates no edge.

Algorithm:

1. **Backtracking Approach**:
   - Start with a vertex and try to build a path by visiting other vertices.
   - For each vertex, check if it is adjacent to the previous vertex and if it has not been visited before.
   - If all vertices are visited and the last vertex is adjacent to the first one, then a Hamiltonian circuit exists.
2. **Menu Options**:
   - Input the adjacency matrix and check if the graph contains a Hamiltonian Circuit.
   - Exit the program.

Source Code:

```
importitertools

defis_hamiltonian(adj, n):
vertices = list(range(n))

   # Fix starting vertex as 0 (avoid duplicate cycles)
for perm in itertools.permutations(vertices[1:]):
path = (0,) + perm

valid = True
for i in range(n-1):
ifadj[path[i]][path[i+1]] == 0:
valid = False
break

      # Check edge from last back to first to complete cycle
if valid and adj[path[-1]][path[0]] == 0:
valid = False
if valid:
return True, path

return False, None
while True:
print("\n----- Hamiltonian Circuit Checker -----")
print("1. Check Hamiltonian Circuit")
print("2. Exit")

choice = int(input("Enter your choice (1-2): "))

if choice == 1:
      n = int(input("Enter number of vertices: "))

print("\nEnter the adjacency matrix:")
adj = [list(map(int, input().split())) for _ in range(n)]

exists, path = is_hamiltonian(adj, n)

if exists:
print("\nResult: Hamiltonian Circuit EXISTS ")
         cycle = " → ".join(str(v) for v in path) + " → " + str(path[0])
print("Hamiltonian Cycle:", cycle)
else:
print("\nResult: Hamiltonian Circuit does NOT Exist ")

elif choice == 2:
print("Exiting program... Thank you!")
break
else:
print("Invalid choice! Please enter 1 or 2.")
```

Output:

```
----- Hamiltonian Circuit Checker -----
1. Check Hamiltonian Circuit
2. Exit
Enter your choice (1-2): 1
Enter number of vertices: 3
Enter the adjacency matrix:
0 1 1
1 0 1
1 1 1
Result: Hamiltonian Circuit EXISTS
Hamiltonian Cycle: 0 → 1 → 2 → 0
----- Hamiltonian Circuit Checker -----
1. Check Hamiltonian Circuit
2. Exit
Enter your choice (1-2): 2
Exiting program... Thank you!
```

Conclusion:

The above program executed successfully

8.Write a python program to demonstrate addition of two binary numbers

Aim: To write a Python program to add two binary numbers entered by the user at runtime and display the result using manual binary addition logic.

Description:

This program performs the addition of two binary numbers entered by the user at runtime. It consists of three main components: input validation, manual binary addition, and output display. First, the program accepts two binary strings from the user. To ensure correctness, a validation function checks whether both inputs contain only binary digits (0 and 1). If an invalid input is detected, an error message is displayed, preventing further execution.

Algorithm:

Step 1: Start the program.

Step 2:Define a function `is_binary(num)` to validate whether the given string is a binary number.

Step 3:Define a function `add_binary(bin1, bin2)` to perform manual binary addition.

Step 4: Inside the main program, accept two binary numbers from the user.

Step 5:Check if both numbers are binary using `is_binary()`.

Step 6: If either number is invalid, print an error message and go to Step 13.

Step 7: Initialize variables:

→ `i = len(bin1) - 1`

→ `j = len(bin2) - 1`

→ `carry = 0`

→ `result = ""`

Step 8: Repeat while `i >= 0` or `j >= 0` or `carry == 1`:

a. If `i >= 0`, set `bit1 = int(bin1[i])`, else `bit1 = 0`

b. If `j >= 0`, set `bit2 = int(bin2[j])`, else `bit2 = 0`

c. Compute `total = bit1 + bit2 + carry`

d. Set `result = str(total % 2) + result`

e. Update `carry = total // 2`

f. Decrement `i` and `j`

Step 9: After the loop ends, `result` contains the final binary sum.

Step 10: Print the original binary numbers.

Step 11:Print the resulting binary sum.

Step 12:Print program completion message.

Step 13: Stop.

Source code:

```
# Function to check if a string is a valid binary number
defis_binary(num):
for char in num:
if char not in ('0', '1'):
return False
return True
# Function to perform manual binary addition
defadd_binary(bin1, bin2):
    i = len(bin1) - 1
    j = len(bin2) - 1
carry = 0
    result = ""
    # Loop through both numbers from right to left
while i >= 0 or j >= 0 or carry == 1:
        bit1 = int(bin1[i]) if i >= 0 else 0
        bit2 = int(bin2[j]) if j >= 0 else 0
        # Binary addition logic
total = bit1 + bit2 + carry
        # Result bit is total % 2
result = str(total % 2) + result
        # Carry is total // 2
carry = total // 2
        i -= 1
        j -= 1
return result
# Main program
print("==== Binary Addition Program ====\n")
# Taking input from user
bin1 = input("Enter first binary number: ")
bin2 = input("Enter second binary number: ")
# Input validation
if not is_binary(bin1) or not is_binary(bin2):
print("\nError: Only binary numbers (0 and 1) are allowed.")
else:
print("\nBoth inputs are valid binary numbers.")
print("Performing binary addition...\n")
    # Calling function to add
result = add_binary(bin1, bin2)
    # Display result
print("First Binary Number   :", bin1)
print("Second Binary Number  :", bin2)
print("Binary Addition Result:", result)
```

Output:

```
Enter first binary number: 1011
Enter second binary number: 11001
Both inputs are valid binary numbers.
Performing binary addition...
First Binary Number   : 1011
Second Binary Number  : 11001
Binary Addition Result: 100100
```

Conclusion:

The above program executed successfully

9. Write a python program to demonstrate subtraction of two binary numbers

Aim: To write a Python program to subtract two binary numbers entered by the user at runtime and display the result using manual binary addition logic.

Description:

This program performs subtraction of two binary numbers entered by the user.It first checks if the inputs contain only 0s and 1s (valid binary numbers).After validation, the program converts both binary numbers to their decimal equivalents, subtracts them, and then converts the result back into binary form.

Algorithm:

1. **Start the program**
2. Display a welcome message.
3. Prompt the user to enter the **first binary number**.
4. Validate the input:
   * If it contains characters other than 0 and 1 → display error and ask again.
5. Prompt the user to enter the **second binary number**.
6. Validate the input similarly.
7. Convert both binary numbers to decimal using:
   ```
   decimal_value = int(binary_string, 2)
   ```
   ```8. Subtract the decimal values:
   ```
   result_decimal = decimal1 - decimal2
   ```
   ```9. Convert the decimal result back to binary:
. * If the result is positive:
   ```
binary_result = bin(result_decimal)[2:]
   ```
   * If the result is negative:
   ```
binary_result = "-" + bin(abs(result_decimal))[2:]
   ```
10. Display:
   * Both input binary numbers
   * Decimal equivalents
11.stop

**Source code:**

```
# Function to check if a string is a valid binary number
defis_binary(num):
    """
Checks if the string contains only 0s and 1s.
    """
forch in num:
ifch not in ('0', '1'):
return False
return True
# Function to get a valid binary number from user
defget_binary_input(prompt):
    """
    Repeatedly asks the user for a binary number until valid.
    """
while True:
value = input(prompt).strip()

if value == "":
print("Input cannot be empty. Try again.\n")
continue

ifis_binary(value):
return value
else:
print("Invalid binary number! Only 0 and 1 are allowed.\n")
# Function to subtract two binary numbers
defsubtract_binary(bin1, bin2):
    """
Subtracts binary numbers using decimal conversion.
    """
    # Convert binary to decimal
    dec1 = int(bin1, 2)
    dec2 = int(bin2, 2)
    # Perform subtraction
result_dec = dec1 - dec2
    # Convert back to binary (handle negative results)
ifresult_dec>= 0:
result_bin = bin(result_dec)[2:]  # remove '0b'
else:
        # negative binary representation
result_bin = "-" + bin(abs(result_dec))[2:]

returnresult_bin, result_dec
# ---------------------- MAIN PROGRAM ---------------------------
print("==================================================")
print("      BINARY SUBTRACTION PROGRAM (LONG VERSION)     ")
print("==================================================\n")
# Get user input
binary1 = get_binary_input("Enter the first binary number: ")
```

```
binary2 = get_binary_input("Enter the second binary number: ")
print("\nCalculating subtraction...")
print("--------------------------------------------------")
# Perform subtraction
binary_result, decimal_result = subtract_binary(binary1, binary2)
# Display output
print(f"\nFirst Binary Number     : {binary1}")
print(f"Second Binary Number    : {binary2}")
print("--------------------------------------------------")
print(f"Decimal Subtraction     : {int(binary1,2)} - {int(binary2,2)} = {decimal_result}")
print(f"Binary Subtraction      : {binary1} - {binary2} = {binary_result}")
print("--------------------------------------------------")
print("\nThank you for using the Binary Subtraction Program!")
```

Output:

```
==================================================
    BINARY SUBTRACTION PROGRAM (LONG VERSION)
==================================================
Enter the first binary number: 1101
Enter the second binary number: 101
Calculating subtraction...
--------------------------------------------------
First Binary Number     : 1101
Second Binary Number    : 101
--------------------------------------------------
Decimal Subtraction     : 13 - 5 = 8
Binary Subtraction      : 1101 - 101 = 1000
--------------------------------------------------
Thank you for using the Binary Subtraction Program!
```

Conclusion:

The above program executed successfully

10. Write a python program to demonstrate multiplication of two binary numbers

Aim: To implement a Python program that takes two binary number multiplication as user input at runtime

Description: This Python program multiplies two binary numbers entered by the user at runtime. The program first validates whether the inputs contain only binary digits (0 and 1).Once both inputs are confirmed valid, they are converted into decimal form using Python's built-in binary conversion capability.

Algorithm:

1. Start the program.
2. Display a heading for the Binary Multiplication Program.
3. Ask the user to input the first binary number.
4. Validate the input:
   * Ensure only 0s and 1s are present.
   * If invalid → display an error and ask again.
5. Ask the user to input the second binary number.
6. Validate the second binary number similarly.
7. Convert both binary numbers to decimal using:

```
decimal = int(binary_string, 2)
```

8. Multiply the two decimal numbers.
9. Convert the multiplication result back to binary using:

```
binary = bin(result)[2:]
```

10. Display the original inputs, decimal equivalents, and multiplication results.
11. End the program.

Source code:

```
# Function to repeatedly get valid binary number input from user
defget_binary_input(message):
    """
    Continuously prompts the user until valid binary input is entered.
    """
while True:
value = input(message).strip()

if value == "":
print("Input cannot be empty. Please try again.\n")
continue

ifis_binary(value):
return value
else:
print("Invalid binary number! Only 0 and 1 are allowed.\n")

# Function to multiply two binary numbers
defmultiply_binary(b1, b2):
    """
    Converts binary strings to decimal, multiplies them,
and converts the result back to binary.
    """
    # Convert binary inputs to decimal
    decimal1 = int(b1, 2)
    decimal2 = int(b2, 2)

    # Perform multiplication
result_decimal = decimal1 * decimal2

    # Convert decimal result back to binary
result_binary = bin(result_decimal)[2:]  # remove '0b' prefix

returnresult_binary, result_decimal
# ------------------------- MAIN PROGRAM -------------------------
print("==================================================")
print("         BINARY MULTIPLICATION PROGRAM (LONG)       ")
print("==================================================\n")
# Take 2 binary numbers from the user at runtime
binary1 = get_binary_input("Enter the first binary number : ")
binary2 = get_binary_input("Enter the second binary number: ")
print("\nProcessing your input...")
print("--------------------------------------------------")
# Perform binary multiplication
binary_result, decimal_result = multiply_binary(binary1, binary2)
# Display results
print("\nFirst Binary Number       :", binary1)
print("Second Binary Number      :", binary2)
print("--------------------------------------------------")
```

```
print(f"Decimal Multiplication    : {int(binary1,2)} × {int(binary2,2)} = {decimal_result}")
print(f"Binary Multiplication     : {binary1} × {binary2} = {binary_result}")
print("--------------------------------------------------")
print("\nThank you for using the Binary Multiplication Program!")
```

Output:

```
==================================================
        BINARY MULTIPLICATION PROGRAM (LONG)
==================================================
Enter the first binary number : 101
Enter the second binary number: 11
Processing your input...
--------------------------------------------------
First Binary Number       : 101
Second Binary Number      : 11
--------------------------------------------------
Decimal Multiplication    : 5 × 3 = 15
Binary Multiplication     : 101 × 11 = 1111
--------------------------------------------------
Thank you for using the Binary Multiplication Program!
```

Conclusion:

The above program executed successfully

11.Write a python program to demonstrate division of two binary numbers

Aim: To develop a Python program that accepts two binary numbers from the user atruntime, performs division of the first binary number by the second

Description: This program performs division of two binary numbers entered by the user during runtime. The program first ensures that both inputs are valid binary numbers (containing only 0s and 1s).

Algorithm:

1. Start the program.
2. Display a heading for the Binary Division Program.
3. Ask the user to input the first binary number.
4. Validate the input:
   * Ensure only digits 0 and 1 are used.
5. Ask the user for the second binary number.
6. Validate the input similarly.
7. Check if the second binary number is **zero**:
   * If yes → show error and ask again.
8. Convert both binary numbers to decimal using:

```
int(binary_string, 2)
```

9. Perform integer division:

```
quotient = decimal1 // decimal2
remainder = decimal1 % decimal2
```

10. Convert both quotient and remainder to binary using:

```
bin(number)[2:]
```

11. Display:

* Binary inputs
* Decimal equivalents
* Quotient and remainder (both decimal and binary)

12. End the program.

Source code:

```
# Function to check whether input is a valid binary number
defis_binary(num):
    """
    Returns True if the string contains only 0 and 1.
    """
for digit in num:
if digit not in ('0', '1'):
return False
return True


# Function to get a valid binary input from user
defget_binary_input(message):
    """
    Keeps asking for input until a valid binary number is entered.
    """
while True:
value = input(message).strip()

if value == "":
print("Input cannot be empty. Please enter a valid binary number.\n")
continue

ifis_binary(value):
return value
else:
print("Invalid binary number! Use digits 0 and 1 only.\n")


# Function to divide two binary numbers
defdivide_binary(b1, b2):
    """
    Converts binary strings to decimal, performs division,
then converts quotient and remainder back to binary.
    """
    decimal1 = int(b1, 2)
    decimal2 = int(b2, 2)

quotient_decimal = decimal1 // decimal2
remainder_decimal = decimal1 % decimal2

    # Convert results to binary (remove '0b' prefix)
quotient_binary = bin(quotient_decimal)[2:]
remainder_binary = bin(remainder_decimal)[2:]

returnquotient_binary, remainder_binary, quotient_decimal, remainder_decimal
```

```
# ------------------------------ MAIN PROGRAM ------------------------------

print("=====================================================")
print("         BINARY DIVISION PROGRAM (LONG VERSION)     ")
print("=====================================================\n")
# Get binary inputs from user at run time
binary1 = get_binary_input("Enter the first binary number  : ")
binary2 = get_binary_input("Enter the second binary number : ")
# Check for division by zero
whileint(binary2, 2) == 0:
print("\nError: Division by ZERO is not allowed!")
   binary2 = get_binary_input("Enter a NON-ZERO binary number: ")
print("\nProcessing your input...")
print("-----------------------------------------------------")

# Perform division
q_bin, r_bin, q_dec, r_dec = divide_binary(binary1, binary2)

# Display results
print("\nFirst Binary Number        :", binary1)
print("Second Binary Number       :", binary2)
print("-----------------------------------------------------")
print(f"Decimal Division           : {int(binary1,2)} ÷ {int(binary2,2)}")
print(f"Quotient (Decimal)         : {q_dec}")
print(f"Remainder (Decimal)        : {r_dec}")
print("-----------------------------------------------------")
print(f"Binary Quotient            : {q_bin}")
print(f"Binary Remainder           : {r_bin}")
print("-----------------------------------------------------")
print("\nThank you for using the Binary Division Program!")

```

Output:

```
==================================================
      BINARY DIVISION PROGRAM (LONG VERSION)
==================================================
Enter the first binary number  : 1101
Enter the second binary number : 11
Processing your input...
--------------------------------------------------
First Binary Number       : 1101
Second Binary Number      : 11
--------------------------------------------------
Decimal Division          : 13 ÷ 3
Quotient (Decimal)        : 4
Remainder (Decimal)       : 1
--------------------------------------------------
Binary Quotient           : 100
Binary Remainder          : 1
--------------------------------------------------

Thank you for using the Binary Division Program!
```

Conclusion:

The above program executed successfully

12.Write a python program to perform decimal,octal,hexa into two binary numbers

Aim: To write a Python program that accepts a Decimal, Octal, or Hexadecimal number as user input at runtime---

Description:

This Python program allows the user to convert a number from three number system Decimal, Octal, or Hexadecimal—into Binary.The user selects the input type (1 for Decimal, 2 for Octal, 3 for Hexadecimal).

Descriprtion:

1. Start the program.
2. Display a menu asking the user to choose the number system:
   * 1 → Decimal
   * 2 → Octal
   * 3 → Hexadecimal
3. Get user choice.
4. Validate the choice.
5. If choice = Decimal:
   * Ask the user to enter a decimal number.
   * Validate all characters are digits 0–9.
   * Convert to binary using `bin(int(value))[2:]`.
6. If choice = Octal:
   * Ask the user to enter an octal number.
   * Validate digits are between 0–7.
   * Convert using `bin(int(value, 8))[2:]`.
7. If choice = Hexadecimal:
   * Ask the user to enter a hex number.
   * Validate characters 0–9 and A–F.
   * Convert using `bin(int(value, 16))[2:]`.
8. Display the binary result.
9. End the program.

Source code:

```
# Function to check if input is a valid decimal number
defis_decimal(num):
returnnum.isdigit()

# Function to check if input is a valid octal number (0-7 only)
defis_octal(num):
for digit in num:
if digit not in "01234567":
return False
return True

# Function to check if input is a valid hexadecimal number
defis_hexadecimal(num):
valid_chars = "0123456789ABCDEFabcdef"
for digit in num:
if digit not in valid_chars:
return False
return True


# ---------------------------- MAIN PROGRAM -----------------------------

print("============================================================")
print("      DECIMAL / OCTAL / HEXADECIMAL TO BINARY CONVERTER   ")
print("============================================================\n")

print("Choose the number system of your input:")
print("1. Decimal")
print("2. Octal")
print("3. Hexadecimal\n")

# Get user choice
choice = input("Enter your choice (1/2/3): ").strip()

while choice not in ('1', '2', '3'):
print("Invalid choice! Please choose 1, 2 or 3.\n")
choice = input("Enter your choice (1/2/3): ").strip()

print()

# DECIMAL INPUT
if choice == '1':
num = input("Enter a decimal number: ").strip()

while not is_decimal(num):
print("Invalid decimal number! Try again.\n")
num = input("Enter a decimal number: ").strip()

    # Conversion
```

```
decimal_value = int(num)
binary_value = bin(decimal_value)[2:]

print("\nDecimal Number       :", num)
print("Binary Equivalent    :", binary_value)


# OCTAL INPUT
elif choice == '2':
num = input("Enter an octal number: ").strip()

while not is_octal(num):
print("Invalid octal number! (Digits 0-7 only)\n")
num = input("Enter an octal number: ").strip()

   # Conversion
decimal_value = int(num, 8)
binary_value = bin(decimal_value)[2:]

print("\nOctal Number         :", num)
print("Binary Equivalent    :", binary_value)


# HEXADECIMAL INPUT
elif choice == '3':
num = input("Enter a hexadecimal number: ").strip()

while not is_hexadecimal(num):
print("Invalid hexadecimal number! Use 0-9 and A-F.\n")
num = input("Enter a hexadecimal number: ").strip()

   # Conversion
decimal_value = int(num, 16)
binary_value = bin(decimal_value)[2:]

print("\nHexadecimal Number   :", num.upper())
print("Binary Equivalent    :", binary_value)

print("\nThank you for using the Number System Converter!")
```

Output:

```
============================================================
      DECIMAL / OCTAL / HEXADECIMAL TO BINARY CONVERTER
============================================================

Choose the number system of your input:
1. Decimal
2. Octal
3. Hexadecimal

Enter your choice (1/2/3): 3

Enter a hexadecimal number: 1F

Hexadecimal Number   : 1F
Binary Equivalent    : 11111

Thank you for using the Number System Converter!
```

Conclusion:

The above program executed successfully

13.Write a Python program that simulates an 8×1 Multiplexer

Aim:
To write a Python program that simulates an 8×1 Multiplexer, where the user enters the 8 input lines (I0–I7) and 3 select lines (S2, S1, S0) at runtime.

Description: An 8-to-1 multiplexer is a digital circuit with eight input lines, three select lines, and one output line. It acts like a switch that directs one of the eight input signals to the single output line, based on the unique binary combination applied to the three select lines.

Algorithm:
1. Start the program.
2. Display a heading for the 8×1 Multiplexer.
3. Ask the user to enter 8 input values (I0 to I7), one by one.
4. Validate that each input is either **0 or 1**.
5. Ask the user to enter the 3 select lines (S2, S1, S0).
6. Validate each select line (must be 0 or 1).
7. Convert the 3 select lines into a decimal number:
   ```
index = S2*4 + S1*2 + S0*1
   ```
8. Select the input value from the list:
   ```
output = inputs[index]
   ```
9. Display the selected input as the multiplexer output.
10. End the program.

Source code:

```
# Function to get a valid binary input (0 or 1)
defget_binary_input(message):
    """
    Repeatedly asks the user for a value until a valid binary digit (0/1)
is entered.
    """
while True:
value = input(message).strip()
if value in ('0', '1'):
returnint(value)
else:
print("Invalid input! Please enter only 0 or 1.\n")


# ---------------------------- MAIN PROGRAM -----------------------------

print("============================================================")
print("                 8 x 1 MULTIPLEXER                  ")
print("============================================================\n")

# Step 1: Read 8 inputs I0 to I7
inputs = []
print("Enter the 8 input values for the multiplexer:\n")

for i in range(8):
inp = get_binary_input(f"Enter I{i} (0/1): ")
inputs.append(inp)

print("\nInputs successfully recorded!")
print("I0-I7 =", inputs, "\n")

# Step 2: Read 3 select lines S2 S1 S0
print("Enter the 3 Select Line values (S2 S1 S0):\n")

S2 = get_binary_input("Enter S2 (0/1): ")
S1 = get_binary_input("Enter S1 (0/1): ")
S0 = get_binary_input("Enter S0 (0/1): ")

# Step 3: Convert select lines to decimal index
index = (S2 * 4) + (S1 * 2) + (S0 * 1)

# Step 4: Determine the output
output = inputs[index]

# Step 5: Display the results
print("\n============================================================")
print("                   MUX OUTPUT                       ")
print("============================================================")
print(f"Select Lines (S2 S1 S0):  {S2}  {S1}  {S0}")
```

```
print(f"Binary Select Value     :  {S2}{S1}{S0}")
print(f"Decimal Select Value    :  {index}")
print("-------------------------------------------------------")
print(f"Selected Input (I{index}): {output}")
print("=======================================================")

print("\nThank you for using the 8x1 Multiplexer Program!")
```

Output:

```
============================================================
                8 x 1 MULTIPLEXER
============================================================

Enter the 8 input values for the multiplexer:

Enter I0 (0/1): 1
Enter I1 (0/1): 0
Enter I2 (0/1): 1
Enter I3 (0/1): 1
Enter I4 (0/1): 0
Enter I5 (0/1): 0
Enter I6 (0/1): 1
Enter I7 (0/1): 0

Inputs successfully recorded!
I0-I7 = [1, 0, 1, 1, 0, 0, 1, 0]

Enter the 3 Select Line values (S2 S1 S0):

Enter S2 (0/1): 1
Enter S1 (0/1): 0
Enter S0 (0/1): 1

============================================================
                  MUX OUTPUT
============================================================
Select Lines (S2 S1 S0):  1  0  1
Binary Select Value     :  101
Decimal Select Value    :  5
------------------------------------------------------------
Selected Input (I5): 0
============================================================
```

Conclusion:

The above program executed successfully

14. Write a python program to perform the working of Shift Register.

Aim: To write a Python program that simulates the working of a Shift Register, where the user enters the number of bits, the initial register values, the shift direction (left/right), and the input bit. The program should perform shifting and show the new register output.

Descrition:
A shift register is a sequential logic circuit that shifts its stored data bits either left or rightwhen a clock pulse occurs.This program allows the user to:
1. Enter the size of the register (number of bits).
2. Enter the initial register content (binary values).
3. Choose the type of shift:
   * Left Shift
   * Right Shift
4. Enter the new input bit (0 or 1).
5. The program performs the shift operation and updates the register.

Algorithm:
1. Start the program.
2. Ask the user for the number of bits in the shift register.
3. Ask the user to enter the initial bit values for the register.
4. Validate each bit (must be 0 or 1).
5. Ask the user to choose:

   * **1 → Left Shift**
   * **2 → Right Shift**
6. Ask the user to enter the new input bit (0 or 1).
7. If **Left Shift**:

   * Remove the **leftmost bit**
   * Insert the **input bit at the right end**
8. If **Right Shift**:

   * Remove the **rightmost bit**
   * Insert the **input bit at the left end**
9. Display:

   * Original register
   * Shift type
   * Input bit
   * Updated register
10. End the program.

Source code:

```
# Function to get a valid binary input (0 or 1)
defget_binary_input(message):
while True:
value = input(message).strip()
if value in ("0", "1"):
returnint(value)
else:
print("Invalid input! Only 0 or 1 allowed.\n")


# ------------------------------ MAIN PROGRAM ------------------------------

print("================================================================")
print("                    SHIFT REGISTER PROGRAM                  ")
print("================================================================\n")

# Step 1: Get register size
while True:
size = input("Enter number of bits in the shift register: ").strip()
ifsize.isdigit() and int(size) > 0:
size = int(size)
break
else:
print("Please enter a valid positive number.\n")

# Step 2: Get initial register content
register = []
print("\nEnter the initial values of the register (0 or 1):")

for i in range(size):
bit = get_binary_input(f"Enter bit {i} : ")
register.append(bit)

print("\nInitial Register State:", register)

# Step 3: Choose shift direction
print("\nChoose Shift Operation:")
print("1. Left Shift")
print("2. Right Shift")

choice = input("Enter your choice (1 or 2): ").strip()

while choice not in ("1", "2"):
print("Invalid choice! Please choose 1 or 2.\n")
choice = input("Enter your choice (1 or 2): ").strip()

# Step 4: Get new input bit
new_bit = get_binary_input("\nEnter the new input bit (0 or 1): ")
```

```
# Step 5: Perform shifting
original_register = register.copy()

if choice == "1":  # Left shift
register.pop(0)          # Remove leftmost bit
register.append(new_bit)   # Add new bit at right
shift_type = "LEFT SHIFT"

else:  # Right shift
register.pop()           # Remove rightmost bit
register.insert(0, new_bit) # Add new bit at left
shift_type = "RIGHT SHIFT"

# Step 6: Output results
print("\n============================================================")
print("                    OUTPUT                    ")
print("============================================================")

print("Shift Type         :", shift_type)
print("Original Register   :", original_register)
print("Input Bit          :", new_bit)
print("New Register State  :", register)

print("\nThank you for using the Shift Register Program!")
```

Output:

```
================================================================
                SHIFT REGISTER PROGRAM
================================================================
Enter number of bits in the shift register: 4

Enter the initial values of the register (0 or 1):
Enter bit 0 : 1
Enter bit 1 : 0
Enter bit 2 : 1
Enter bit 3 : 1

Initial Register State: [1, 0, 1, 1]

Choose Shift Operation:
1. Left Shift
2. Right Shift
Enter your choice (1 or 2): 1

Enter the new input bit (0 or 1): 0

================================================================
                  OUTPUT
================================================================
Shift Type        : LEFT SHIFT
Original Register  : [1, 0, 1, 1]
Input Bit         : 0
New Register State  : [0, 1, 1, 0]

Thank you for using the Shift Register Program!
```

Conclusion:

The above program executed successfully

15. Write a python program that simulates 3X8 Decoder.

Aim: To write a Python program that simulates the behavior of a 3×8 Decoder, where the user enters 3 input bits at runtime and the program activates exactly one of the eight output lines (Y0–Y7) based on the binary input combination.

Description:
A **3×8 decoder** is a combinational logic circuit that converts **3 input bits** (A2, A1, A0) into **8 unique output lines** (Y0–Y7).
Only ONE output becomes **1**, and the rest stay **0**, based on the binary value of the inputs.

Algorithm:
1. Start the program.
2. Ask the user to enter the **three input bits** (A2, A1, A0).
3. Validate each input bit (must be 0 or 1).
4. Convert the input bits into a decimal number:
   ```
index = A2*4 + A1*2 + A0
   ```
5. Create a list of eight outputs initialized to 0:
   ```
outputs = [0,0,0,0,0,0,0,0]
   ```
6. Set the selected output line to 1:
   ```
outputs[index] = 1
   ```
7. Display:

   * Input values
   * Decimal equivalent
   * Activated output line
   * Full output list (Y0–Y7)
8. End the program.

Source code:

```
# Function to get a valid binary input (0 or 1)
defget_binary_input(message):
while True:
value = input(message).strip()
if value in ("0", "1"):
returnint(value)
else:
print("Invalid input! Please enter only 0 or 1.\n")

# ------------------------------ MAIN PROGRAM --------------------------

print("============================================================")
print("                  3 x 8 DECODER PROGRAM                ")
print("============================================================\n")

# Step 1: Take the 3 input bits
A2 = get_binary_input("Enter A2 (MSB) (0/1): ")
A1 = get_binary_input("Enter A1 (0/1): ")
A0 = get_binary_input("Enter A0 (LSB) (0/1): ")

# Step 2: Convert inputs to decimal index
index = (A2 * 4) + (A1 * 2) + (A0 * 1)

# Step 3: Create output list and activate one output
outputs = [0] * 8   # Y0, Y1, ..., Y7
outputs[index] = 1   # Activate selected output

# Step 4: Display results
print("\n============================================================")
print("                      OUTPUT                          ")
print("============================================================")
print(f"Input Bits (A2 A1 A0):  {A2} {A1} {A0}")
print(f"Binary Input Value   :  {A2}{A1}{A0}")
print(f"Decimal Equivalent   :  {index}")
print("------------------------------------------------------------")
print(f"Activated Output Line:  Y{index}")
print("------------------------------------------------------------")
print("Decoder Output (Y0–Y7): ", outputs)
print("============================================================\n")

print("Thank you for using the 3×8 Decoder Program!")
```

Output:

```
============================================================
                3 x 8 DECODER PROGRAM
============================================================

Enter A2 (MSB) (0/1): 1
Enter A1 (0/1): 0
Enter A0 (LSB) (0/1): 1

============================================================
                      OUTPUT
============================================================
Input Bits (A2 A1 A0):  1 0 1
Binary Input Value   :  101
Decimal Equivalent   :  5
------------------------------------------------------------
Activated Output Line:  Y5
------------------------------------------------------------
Decoder Output (Y0–Y7):  [0, 0, 0, 0, 0, 1, 0, 0]
============================================================
```

Conclusion:

The above program executed successfully