LAB_2\chinese_remainder_theorem.py

```python

# 21CSE1003
# Ashish Singh
# LAB 2

# Q1. Write a program to Implement Chinese Reminder Algorithm.

from math import gcd

def coprime(l: list):
    for i in range(0, len(l)):
        for j in range(i+1, len(l)):
            if gcd(l[i], l[j]) != 1:
                return False
    return True

def chinese_remainder(a: list, m: list):
    if coprime(m) == False:
        print("\nError: The moduli must be pairwise coprime.")
        return
    M = 1
    for i in m:
        M *= i
    Mi = []
    for i in m:
        Mi.append(M//i)
    yi = [] # this is Mi inverse
    for i in range(0, len(m)):
        yi.append(pow(Mi[i], -1, m[i]))
    x = 0
    for i in range(0, len(m)):
        x += a[i]*Mi[i]*yi[i]
    x = x % M
    print(f"\nThe solution is x = {x}\n")

n = int(input("\nEnter the number of equations: "))
a = []
m = []
for i in range(0, n):
    a.append(int(input(f"Enter a{i+1}: ")))
    m.append(int(input(f"Enter m{i+1}: ")))

chinese_remainder(a, m)

```

LAB_2\modular_inverse_of_matrix.py

```python

# Q2. Write a program to find the Modular Inverse of a 3x3 matrix using Extended Euclidian Algorithm.

import numpy as np

def determinant(matrix):
    return np.linalg.det(matrix)

def modular_inverse(m, n):
    actual_det = determinant(m)
    if determinant(m) == 0:
        print("\nDeterminant does not exist.")
        return
    if determinant(m) < 0:
        det = round(determinant(m) % n)
    else:
        det = round(determinant(m))

    inv = np.linalg.inv(m)
    inv = inv * actual_det
    print("\nInverse of the matrix is:")
    print(f"1/{det} *\t {inv}")

    c = 1 # c is mulitpicative inverse
    while (det * c) % n != 1:
        c += 1

    print(f"\nMultiplicative inverse of {det}^-1 is: {c}")

    print(f"\n{c} * {inv} mod {n}")
    inv = c * inv
    print(f"\n{inv} mod {n}")

    mod_inv = np.mod(inv, 26)

    print(f"\nModular inverse of matrix is: \n{mod_inv}")

rows = int(input("Enter the number of rows: "))
cols = int(input("Enter the number of columns: "))
print("Enter the matrix values separated by space followed by newline:")

matrix = np.array([input().strip().split() for _ in range(rows)], int)

n = int(input("\nEnter n: "))

modular_inverse(matrix, n)

```

LAB_2\modular_expnentiation.py

```python

# Q3. Write a program to implement Modular exponentiation using repeated square and multiply algorithm.

def modular_exponentiation(base, exponent, modulus):
    result = 1
    base = base % modulus

    while exponent > 0:
        if exponent % 2 == 1:
            result = (result * base) % modulus
        exponent = exponent // 2
        base = (base * base) % modulus

    return result

base = int(input("\nEnter the base value: "))
exponent = int(input("Enter the exponent value: "))
modulus = int(input("Enter the value of n: "))

result = modular_exponentiation(base, exponent, modulus)

print(f"Ans = {result}\n")
```

LAB_2\euler_totient_in_Zn.py

```python

# Q4. Write a program to find Euler totient value in Zn.

from math import gcd

def euler_totient(n):
    Zn = [i for i in range(n)]
    print(f"Zn = {Zn}")

    Zn_ = [] # this is Zn*
    for i in Zn:
        if gcd(i, n) == 1:
            Zn_.append(i)
    print(f"Zn* = {Zn_}")

    phi = len(Zn_)
    print(f"\nphi({n}) = {phi}")

n = int(input("\nEnter value of n: "))

euler_totient(n)
```

LAB_2\order_of_Zn.py

```python

# Q5. Write a program to find order of modulo n in Zn.

from math import gcd

def order_of_Zn(n):
    Zn = [i for i in range(n)]
    Zn = set(Zn)
    print(f"Zn = {Zn}")

    Zn_ = set() # this is Zn*
    for i in Zn:
        if gcd(i, n) == 1:
            Zn_.add(i)
    print(f"Zn* = {Zn_}")

    phi = len(Zn_)
    print(f"\nphi({n}) = {phi}")

    t = set() # factors of phi(n)
    for x in range(1, phi+1):
        if phi % x == 0:
            t.add(x)
    print(f"\nfactors of phi({n}) = {t}")

    order = set()

    for i in Zn_:
        for j in t:
            if i**j % n == 1:
                order.add((i, j))
                break
    print(f"\norder of Zn = {order}\n")

n = int(input("\nEnter the value of n: "))
order_of_Zn(n)

```