Practical 3\practical.py

```python
######################################################################
                        #Practical 3A
######################################################################
def EEA(a: int ,b: int) -> tuple:
    r1: int = a
    r2: int = b
    t1: int = 0
    t2: int = 1
    s1: int = 1
    s2: int = 0
    print('-'*70)
    print(f'|{'q':^6}|',end="")
    print(f'{'r1':^6}|',end="")
    print(f'{'r2':^6}|',end="")
    print(f'{'r':^6}|',end="")
    print(f'{'t1':^6}|',end="")
    print(f'{'t2':^6}|',end="")
    print(f'{'t':^6}|',end="")
    print(f'{'s1':^6}|',end="")
    print(f'{'s2':^6}|',end="")
    print(f'{'s':^6}|')
    print('-'*70)

    while(r2 > 0):
        q: int = r1 // r2
        print(f'|{q:^6}|',end="")
        print(f'{r1:^6}|',end="")
        print(f'{r2:^6}|',end="")
        r: int = r1 - q * r2
        r1, r2 = r2, r
        print(f'{r:^6}|',end="")

        print(f'{t1:^6}|',end="")
        print(f'{t2:^6}|',end="")
        t: int = t1 - q * t2
        t1, t2 = t2, t
        print(f'{t:^6}|',end="")

        print(f'{s1:^6}|',end="")
        print(f'{s2:^6}|',end="")
        s: int = s1 - q * s2
        s1, s2 = s2, s
        print(f'{s:^6}|')
    print(f'|{"":^6}|',end="")
    print(f'{r1:^6}|',end="")
    print(f'{r2:^6}|',end="")
    print(f'{"":^6}|',end="")
    print(f'{t1:^6}|',end="")
    print(f'{t2:^6}|',end="")
    print(f'{"":^6}|',end="")
    print(f'{s1:^6}|',end="")
    print(f'{s2:^6}|',end="")
```

```python
    print(f'{"":^6}|')

    print('-'*70)

    return (r1, t1, s1)

def main() -> None:
    print(f'{'start':-^40}')
    a: int = int(input("Enter A: "))
    b: int = int(input("Enter B: "))

    result = EEA(a=a,b=b)
    print(f"GCD({a},{b}) = {result[0]}")
    print("coefficients of Bezout's")
    print("t:",result[1])
    print("s:",result[2])
    print(f'{'end':-^40}')

if __name__ == '__main__':
    main()
######################################################################
                        #Practical 3B
######################################################################
def EEA(a: int, b: int) -> int:
    r1: int = a
    r2: int = b
    t1: int = 0
    t2: int = 1
    while r2 > 0:
        q: int = r1 // r2
        r: int = r1 - q * r2
        r1, r2 = r2, r
        t: int = t1 - q * t2
        t1, t2 = t2, t
    return t1


def multiplicativeCipher(inputText: str, key: int):
    alphabet: str = "abcdefghijklmnopqrstuvwxyz"
    result: str = ""
    if inputText.islower():
        for ch in inputText:
            result += alphabet[(alphabet.find(ch) * key) % 26].upper()
        print("Encrypted:", result)
    else:
        t = EEA(26, key)
        while t < 0:
            t += 26
        for ch in inputText.lower():
            result += alphabet[(alphabet.find(ch) * t) % 26].lower()
        print("Decrypted:", result)

def AffineCipher(inputText: str, k1: int, k2: int):
    alphabet: str = "abcdefghijklmnopqrstuvwxyz"
```

```
    result: str = ""
    if inputText.islower():
        for ch in inputText:
            result += alphabet[((alphabet.find(ch) * k1) + k2) % 26].upper()
        print("Encrypted:", result)
    else:
        t = EEA(26, k1)
        while t < 0:
            t += 26
        for ch in inputText.lower():
            index: int = alphabet.find(ch) - k2
            while( index < 0):
                index += 26
            result += alphabet[(index * t) % 26].lower()
        print("Decrypted:", result)


def main() -> None:
    print(f'{'start':-^40}')

    choice: str = input('''
    Enter
    (M) for Multiplicative Cipher
    (A) for Affine Cipher
    Enter Your Choice: ''').lower()
    match choice:
        case 'm':
            print('-'*40)
            inputText: str = input("Enter Your Text: ")
            key: int = int(input("Enter Your Key: "))
            print('-'*40)
            multiplicativeCipher(inputText, key)
        case 'a':
            print('-'*40)
            inputText: str = input("Enter Your Text: ")
            k1: int = int(input("Enter Key1: "))
            k2: int = int(input("Enter Key2: "))
            print('-'*40)
            AffineCipher(inputText, k1, k2)

    print(f'{'end':-^40}')


if __name__ == "__main__":
    main()
##################################################################
                        #Practical 3C
##################################################################
import random

def EEA(a: int, b: int) -> int:
    r1: int = a
    r2: int = b
    t1: int = 0
```

```
    t2: int = 1
    while r2 > 0:
        q: int = r1 // r2
        r: int = r1 - q * r2
        r1, r2 = r2, r
        t: int = t1 - q * t2
        t1, t2 = t2, t
    return t1

def hillCipher(text: str, matrix: list[int]):
    size: int = 2
    alphabet: str = 'abcdefghijklmnopqrstuvwxyz'
    flag: bool = False if text.islower() else True

    for i, v in enumerate(matrix):
        while v < 0:
            v += 26
        matrix[i] = v

    if flag:
        text = text.lower()
        a, b, c, d = matrix
        det = ((a*d) - (b*c))
        while( det < 0):
            det += 26
        detinv = EEA(26, det)
        while(detinv < 0):
            detinv += 26
        adjoint = [d, -b, -c, a]
        for i, v in enumerate(adjoint):
            while v < 0:
                v += 26
            adjoint[i] = v
        for i, v in enumerate(adjoint):
            matrix[i] = v * detinv


    while(len(text) % size):
        text += random.choice("".join(ch for ch in alphabet if ch not in
[*text]))

    textlist: list[str] = [text[i:i+size] for i in range(0,len(text),size)]

    result: str = ""
    for txt in textlist:
        plist: list[int] = list()
        for ch in txt:
            plist.append(alphabet.find(ch))
        x, y = plist
        a, b, c, d = matrix
        clist: list[int] = [(a*x + c*y)%26, (b*x + d*y)%26]
        for v in clist:
            result += alphabet[v]

```

```
    if flag:
        print("Decrypted:", result.lower())
    else:
        print("Encrypted:", result.upper())


def main() -> None:
    print(f'{'start':-^40}')
    matrix: list[int] = list(map(int, input("Enter Your Matrix a, b, c, d:
").strip().split(" ")))
    inputText: str = input("Enter Your String: ")
    print('-'*40)
    hillCipher(inputText, matrix)
    print(f'{'end':-^40}')

if __name__ == '__main__':
    main()
```