第三次作业：

作业1

使用面向对象的编程实现凯撒加密解密，将一个长度不小于10 的英文字符串输入，用户选择任意一个整数K，实现将英文字符串原文中的每个字母(例如A)替换为26个英文字母中A之后的第K个位置的字母(例如K=2 时A替换为C)，对标点和空格不更改；随后对加密后的信息进行解密。

具体要求：

1. 从键盘输入整数K，对不合法的输入进行处理。输入K后，程序自动显示加密及解密结果。
2. 要求能够处理字母替换的循环（如z+1=a），以及大小写情况。

作业2

使用面向对象的编程实现维吉尼亚密码加密解密，将输一个长度不小于10的英文字符串输入，26行字母表每一行都由前一行向左偏移一位来构成维吉尼亚表格，选择某一关键词并重复而得到密钥，如关键词GOOD，则密钥为GOODGOODGO，对标点和空格不更改；随后对加密后的信息进行解密。

具体要求：

从键盘输入密钥，对不合法的输入进行处理。输入密钥后，以Enter键或Space键作为输入结束标志。程序自动显示加密及解密结果。

维吉尼亚密码原理可参考：百度百科：维吉尼亚密码

代码

```python
"""
* coding=utf-8
* python=3.11
* File: Experiment3.py
* Author: donghy23@mails.tsinghua.edu.cn
* Created: 2024-7-12
* Repo: https://github.com/FHYQ-Dong/Tsinghua-Program-Design-Assignment-3/blob/main/Experiment3/Experiment3.py
"""

import argparse


class CaesarCrypto():
    """
    The CaesarCrypto class represents a Caesar cipher encryption and decryption algorithm.

    ### Args:
        plaintext (str, optional): The plaintext message to be encrypted. Defaults to None.
        key (int, optional): The encryption key. Defaults to None.
        ciphertext (str, optional): The ciphertext message to be decrypted. Defaults to None.
    ### Raises:
        ValueError: If the key is not provided.

    ### Attributes:
        plaintext (str): The plaintext message.
        key (int): The encryption key.
        ciphertext (str): The ciphertext message.
    """
    def __init__(
        self,
        plaintext: str|None = None,
        key: int|None = None,
        ciphertext: str|None = None
    ):
        self._plaintext = None
        self._key = None
        self._ciphertext = None

        if not key:
            raise ValueError("Key must be provided")
        else:
            self._key = key
            if plaintext and not ciphertext:
                self._plaintext = plaintext
                self._ciphertext = ciphertext
                self.encrypt()
            elif not plaintext and ciphertext:
                self._plaintext = plaintext
                self._ciphertext = ciphertext
                self.decrypt()
            elif plaintext and ciphertext:
                if not self.valid():
                    raise ValueError("Plain text, key and cipher text not
match")
                else:
                    self._plaintext = plaintext
                    self._ciphertext = ciphertext

    def encrypt(self):
        """
        Encrypts the plaintext message using the Caesar cipher algorithm.
        """
        self._ciphertext = ''
        k = self._key % 26
        for c in self._plaintext:
            if 'a' <= c <= 'z':
                self._ciphertext += chr(
                    ord(c) + k if ord(c) + k <= ord('z') \
                    else ord(c) + k - ord('z') + ord('a') - 1
                )
            elif 'A' <= c <= 'Z':
                self._ciphertext += chr(
                    ord(c) + k if ord(c) + k <= ord('Z') \
                    else ord(c) + k - ord('Z') + ord('A') - 1
                )
            else:
                self._ciphertext += c

    def decrypt(self):
        """
        Decrypts the ciphertext message using the Caesar cipher algorithm.
        """
        self._plaintext = ''
        k = self._key % 26
        for c in self._ciphertext:
            if 'a' <= c <= 'z':
                self._plaintext += chr(
                    ord(c) - k if ord(c) - k >= ord('a') \
                    else ord('z') - ord('a') + ord(c) - k + 1
                )
            elif 'A' <= c <= 'Z':
                self._plaintext += chr(
                    ord(c) - k if ord(c) - k >= ord('A') \
                    else ord('Z') - ord('A') + ord(c) - k + 1
                )
            else:
                self._plaintext += c

    def valid(self) -> bool:
        """
        Checks if the plaintext, key, and ciphertext match.

        ### Returns:
            bool: True if the plaintext, key, and ciphertext match, False
otherwise.
        """
        temp = ''
        k = self._key % 26
        for c in self._plaintext:
            if 'a' <= c <= 'z':
                temp += chr(
                    tmp := ord(c) + k if tmp <= ord('z') \
                    else ord(c) + k - ord('z') + ord('a') - 1
                )
            elif 'A' <= c <= 'Z':
                temp += chr(
                    tmp := ord(c) + k if tmp <= ord('Z') \
                    else ord(c) + k - ord('Z') + ord('A') - 1
                )
            else:
                temp += c
        return temp == self._ciphertext

    @property
    def plaintext(self):
        return self._plaintext

    @property
    def key(self):
```

```python
        return self._key

    @property
    def ciphertext(self):
        return self._ciphertext

    @plaintext.setter
    def plaintext(self, plaintext):
        self._plaintext = plaintext
        self.encrypt()

    @ciphertext.setter
    def ciphertext(self, ciphertext):
        self._ciphertext = ciphertext
        self.decrypt()


class VigenereCrypto():
    """
    VigenereCrypto class represents a Vigenere cipher encryption and decryption algorithm.

    ### Args:
        plaintext (str|None): The plaintext to be encrypted. Default is None.
        key (str|None): The encryption key. Default is None.
        ciphertext (str|None): The ciphertext to be decrypted. Default is None.

    ### Raises:
        ValueError: If the key is not provided.

    ### Attributes:
        plaintext (str): The plaintext.
        key (str): The encryption key.
        ciphertext (str): The ciphertext.

    """
    def __init__(
        self,
        plaintext: str|None = None,
        key: str|None = None,
        ciphertext: str|None = None
    ):
        self._plaintext = None
        self._key = None
        self._ciphertext = None
        self._encrypt_dict = [
            [chr((i + j) % 26 + ord('a')) for j in range(26)] for i in range(26)
        ]
        self._decrypt_dict = [
            [chr((i - j) % 26 + ord('a')) for j in range(26)] for i in range(26)
        ]

        if not key:
            raise ValueError("Key must be provided")
        else:
            self._key = key.lower()
            if plaintext and not ciphertext:
                self._plaintext = plaintext
                self._ciphertext = ciphertext
                self.encrypt()
            elif not plaintext and ciphertext:
                self._plaintext = plaintext
                self._ciphertext = ciphertext
                self.decrypt()
            elif plaintext and ciphertext:
                if not self.valid():
                    raise ValueError("Plain text, key and cipher text not 
match")
                else:
                    self._plaintext = plaintext
                    self._ciphertext = ciphertext

    def _encrypt_char(self, c: str, k: str) -> str:
        """
        Encrypts a single character using the Vigenere cipher algorithm.

        ### Args:
            c (str): The character to be encrypted.
            k (str): The key character.

        ### Returns:
            str: The encrypted character.
        """
        return self._encrypt_dict[ord(k) - ord('a')][ord(c) - ord('a')]

    def _decrypt_char(self, c: str, k: str) -> str:
        """
        Decrypts a single character using the Vigenere cipher algorithm.

        ### Args:
            c (str): The character to be decrypted.
            k (str): The key character.

        ### Returns:
            str: The decrypted character.
        """
        return self._decrypt_dict[ord(c) - ord('a')][ord(k) - ord('a')]

    def encrypt(self):
        """
        Encrypts the plaintext using the Vigenere cipher algorithm.
        """
        self._ciphertext = ''
        for i, c in enumerate(self._plaintext):
            k = self._key[i % len(self._key)]
            if 'a' <= c <= 'z':
                self._ciphertext += self._encrypt_char(c, k)
            elif 'A' <= c <= 'Z':
                self._ciphertext += self._encrypt_char(c.lower(), k).upper()
            else:
                self._ciphertext += c


    def decrypt(self):
        """
        Decrypts the ciphertext using the Vigenere cipher algorithm.
        """
        self._plaintext = ''
        for i, c in enumerate(self._ciphertext):
            k = self._key[i % len(self._key)]
            if 'a' <= c <= 'z':
                self._plaintext += self._decrypt_char(c, k)
            elif 'A' <= c <= 'Z':
                self._plaintext += self._decrypt_char(c.lower(), k).upper()
            else:
                self._plaintext += c

    def valid(self) -> bool:
        """
        Checks if the plaintext, key, and ciphertext match.

        ### Returns:
            bool: True if the plaintext, key, and ciphertext match, False
otherwise.
        """
        temp = ''
        k = self._key
        for i, c in enumerate(self._plaintext):
            if 'a' <= c <= 'z':
                temp += chr(
                    tmp := ord(c) + ord(k[i % len(k)]) - ord('a') if tmp <=
ord('z') \
                    else ord(c) + ord(k[i % len(k)]) - ord('a') - ord('z') +
ord('a') - 1
                )
            elif 'A' <= c <= 'Z':
                temp += chr(
                    tmp := ord(c) + ord(k[i % len(k)]) - ord('A') if tmp <=
ord('Z') \
                    else ord(c) + ord(k[i % len(k)]) - ord('A') - ord('Z') +
ord('A') - 1
                )
            else:
                temp += c
        return temp == self._ciphertext

    @property
    def plaintext(self):
        """
        str: The plaintext.
        """
        return self._plaintext

    @property
    def key(self):
```

```python
        """
        str: The encryption key.
        """
        return self._key

    @property
    def ciphertext(self):
        """
        str: The ciphertext.
        """
        return self._ciphertext

    @plaintext.setter
    def plaintext(self, plaintext):
        self._plaintext = plaintext
        self.encrypt()

    @ciphertext.setter
    def ciphertext(self, ciphertext):
        self._ciphertext = ciphertext
        self.decrypt()

    @key.setter
    def key(self, key):
        self._key = key.lower()
        self.encrypt()
        self.decrypt()


def Caesar_test():
    pt = "XYZAB,xyzab"
    ky = 28
    cc = CaesarCrypto(pt, ky, None)
    ct = cc.ciphertext
    print(f"------- Caesar Encrypt Test -------\nPlain text: {pt}\nKey: {ky}\nCipher text: {ct}")

    cc = CaesarCrypto(None, ky, ct)
    pt = cc.plaintext
    print(f"------- Caesar Decrypt Test -------\nCipher text: {ct}\nKey: {ky}\nPlain text: {pt}")


def Vigenere_test():
    pt = "XYZAB,xyzab"
    ky = "KEY"
    cc = VigenereCrypto(pt, ky, None)
    ct = cc.ciphertext
    print(f"------- Vigenere Encrypt Test -------\nPlain text: {pt}\nKey: {ky}\nCipher text: {ct}")

    cc = VigenereCrypto(None, ky, ct)
    pt = cc.plaintext
    print(f"------- Vigenere Decrypt Test -------\nCipher text: {ct}\nKey: {ky}\nPlain text: {pt}")
```

```python
def main():
    """
    Main function for the Crypto program.

    This function parses the command line arguments and performs the specified operation
    based on the provided method and mode.

    ### Args:
        --method (str): The encryption method to use. Valid options are "caesar" or "vigenere".
        --mode (str): The operation mode. Valid options are "encrypt", "decrypt", or "test".
        -k, --key (int): The key to use for encryption or decryption.
        -p, --plaintext (str): The plain text to encrypt.
        -c, --ciphertext (str): The cipher text to decrypt.

    ### Returns:
        None
    """
    parser = argparse.ArgumentParser("Crypto")
    parser.add_argument("--method", type=str, help="[caesar|vigenere]")
    parser.add_argument("--mode", type=str, help="[encrypt|decrypt|test]")
    parser.add_argument("-k", "--key", type=int, help="Key for encrypt/decrypt")
    parser.add_argument("-p", "--plaintext", type=str, help="Plain text")
    parser.add_argument("-c", "--ciphertext", type=str, help="Cipher text")
    args = parser.parse_args()

    if args.mode == "test":
        if args.method == "caesar":
            Caesar_test()
        elif args.method == "vigenere":
            Vigenere_test()
        else:
            parser.print_help()
        return

    elif args.mode == "encrypt":
        if args.method == "caesar":
            try:
                cc = CaesarCrypto(args.plaintext, args.key, None)
                print(f"Encrypt...\nPlain text: {args.plaintext}\nKey: {args.key}\nCipher text: {cc.ciphertext}")
            except ValueError as e:
                print(e)
                parser.print_help()
        elif args.method == "vigenere":
            try:
                cc = VigenereCrypto(args.plaintext, args.key, None)
                print(f"Encrypt...\nPlain text: {args.plaintext}\nKey: {args.key}\nCipher text: {cc.ciphertext}")
            except ValueError as e:
                print(e)
                parser.print_help()
        else:
            parser.print_help()

    elif args.mode == "decrypt" :
        if args.method == "caesar":
            try:
                cc = CaesarCrypto(None, args.key, args.ciphertext)
                print(f"Decrypt...\nPlain text: {cc.plaintext}\nKey:
{args.key}\nCipher text: {args.ciphertext}")
            except ValueError as e:
                print(e)
                parser.print_help()
        elif args.method == "vigenere":
            try:
                cc = VigenereCrypto(None, args.key, args.ciphertext)
                print(f"Decrypt...\nPlain text: {cc.plaintext}\nKey:
{args.key}\nCipher text: {args.ciphertext}")
            except ValueError as e:
                print(e)
                parser.print_help()
        else:
            parser.print_help()

    else:
        parser.print_help()


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

结果

输入1:

```
python .\Experiment3.py --method caesar --mode test
```

输出1:

```
------- Caesar Encrypt Test -------
Plain text: XYZAB,xyzab
Key: 28
Cipher text: ZABCD,zabcd
------- Caesar Decrypt Test -------
Cipher text: ZABCD,zabcd
Key: 28
Plain text: XYZAB,xyzab
```

输入2:

```
python .\Experiment3.py --method vigenere --mode test
```

输出2：

```
------- Vigenere Encrypt Test -------
Plain text: XYZAB,xyzab
Key: KEY
Cipher text: HCXKF,hcxkf
------- Vigenere Decrypt Test -------
Cipher text: HCXKF,hcxkf
Key: KEY
Plain text: XYZAB,xyzab
```

输入3：

```
python .\Experiment3.py --method caesar --mode encrypt -p abced123,XYZ -k 123
```

输出3：

```
Encrypt...
Plain text: abced123,XYZ
Key: 123
Cipher text: tuvxw123,QRS
```

输入4：

```
python .\Experiment3.py --method vigenere --mode encrypt -p abced123,XYZ -k abcde
```

输出4：

```
Encrypt...
Plain text: abced123,XYZ
Key: abcde
Cipher text: acehh123,BYA
```