MINISTRY OF EDUCATION, CULTURE AND RESEARCH OF THE REPUBLIC OF MOLDOVA

Technical University of Moldova Faculty of Computers, Informatics and Microelectronics Department of Software and Automation Engineering

Miricinschi Gabriel FAF-233

Report

Laboratory work №1

of Cryptography

Checked by:
Maia Zaica, *university assistant*
FCIM, UTM

Chișinău – 2025

Purpose of Laboratory Work

The purpose of this work is to study the classical Caesar cipher, applying encryption and decryption with a single shift key, and to extend the method by experimenting with a two-key variation in order to compare their security and effectiveness.

Conditions of the Problem

1. Implement the Caesar cipher with key 1–25. Input: only A–Z/a–z, otherwise show warning. Text is uppercased and spaces removed. User selects encrypt/decrypt, enters key and text, and gets the result.

2. Implement the Caesar cipher with two keys, keeping conditions from Task 1.1. Key 2 must use only Latin letters and have length of at least 7.

Technical Implementation

The implementation includes two main Caesar cipher variants:

Simple Caesar Cipher: Shifts letters by a numeric key (1–25). Input text is converted to uppercase, and spaces are removed in the output.

Keyword Caesar Cipher: Builds a custom alphabet by removing duplicate letters from a keyword and appending the remaining letters. This version preserves letter case and leaves non-alphabetic characters unchanged.

A menu-based interface allows users to encrypt, decrypt, and analyze cryptograms with either method. Input validation restricts numeric keys to 1–25 and keywords to letters only, with a maximum length of 7.

Core functions include modular character shifting for alphabet wrapping, keyword duplicate filtering, custom alphabet generation, and cryptogram analysis that outputs all possible decryptions.

1. Simple Caesar Encryption

Encrypts text using a single numeric key. Each letter is shifted forward by `key` positions.

```
string caesarEncrypt(const string &text, int key)
{
```

```
    string result;
    for (char c: text)
    {
        int index = ALPHABET.find(c);
        int newIndex = (index + key) % ALPHABET_SIZE;
        result.push_back(ALPHABET[newIndex]);
    }
    return result;
}
```

2. Simple Caesar Decryption

Decrypts text encrypted with a single numeric key. Each letter is shifted backward by `key` positions.

```
string caesarDecrypt(const string &text, int key)
{
    string result;
    for (char c: text)
    {
        int index = ALPHABET.find(c);
        int newIndex = (index - key + ALPHABET_SIZE) %
           ALPHABET_SIZE;
        result.push_back(ALPHABET[newIndex]);
    }
    return result;
}
```

3. Two-Key Caesar Encryption

Encrypts text using a numeric key and a keyword. A custom alphabet is generated from the keyword to perform the shift.

```
string caesar2KeyEncrypt(const string &text, int key1,
   const string &key2)
{
    string alphabet = permuteAlphabet(key2);
    string result;
```

```
    for (char c: text)
    {
        int index = alphabet.find(c);
        int newIndex = (index + key1) % ALPHABET_SIZE;
        result.push_back(alphabet[newIndex]);
    }
    return result;
}
```

4. Two-Key Caesar Decryption

Decrypts text encrypted with the two-key Caesar cipher. Reverses the numeric shift using the same custom alphabet.

```
string caesar2KeyDecrypt(const string &text, int key1,
   const string &key2)
{
    string alphabet = permuteAlphabet(preprocessText(key2)
       );
    string result;
    for (char c: text)
    {
        int index = alphabet.find(c);
        int newIndex = (index - key1 + ALPHABET_SIZE) %
           ALPHABET_SIZE;
        result.push_back(alphabet[newIndex]);
    }
    return result;
}
```

5. Text Preprocessing

Converts input text to uppercase and removes non-alphabetic characters and spaces.

```
string preprocessText(string text)
{
    string result;
    for (char c: text)
```

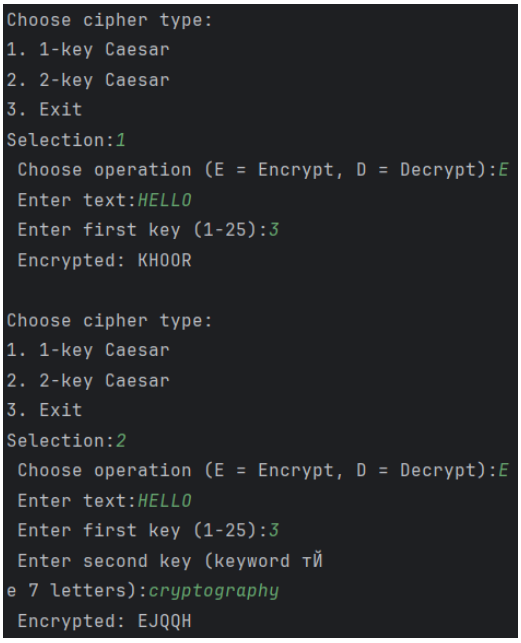
```
    {
        if (isalpha(c))
        {
            result.push_back(toupper(c));
        }
    }
    return result;
}
```

6. Custom Alphabet Generation

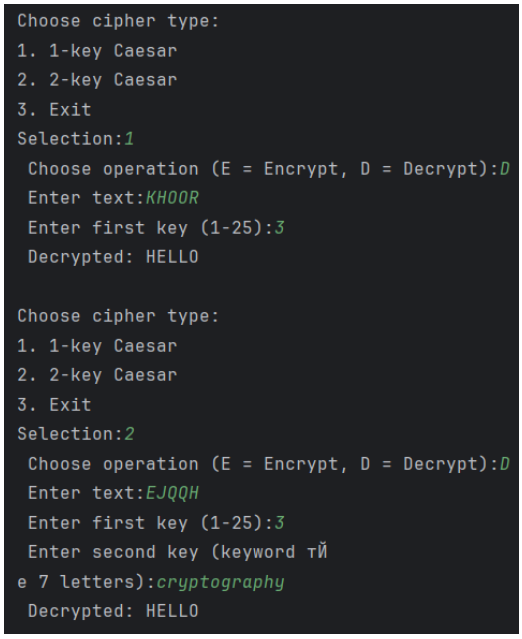
Creates a new alphabet for the two-key cipher by placing the unique letters of the keyword first, followed by the remaining letters of the standard alphabet.

```
string permuteAlphabet(const string &key)
{
    string result = removeDuplicates(preprocessText(key));
    for (char c: ALPHABET)
    {
        if (result.find(c) == string::npos)
        {
            result.push_back(toupper(c));
        }
    }
    return result;
}
```

Visual Representation



(a) Encrypt



(b) Decrypt

Figure 1: Screenshots of the program output from the console

Conclusions

The implementation of the Caesar cipher with both single-key and two-key variants demonstrates the versatility of classical encryption methods. By validating inputs, preprocessing text, and allowing dynamic key selection, the program ensures reliable encryption and decryption. The single-key version highlights the simplicity of basic substitution, while the two-key extension introduces custom alphabets for added complexity. Together, these implementations provide a clear understanding of substitution ciphers and their role as a foundation for more advanced cryptographic techniques.