**Ministerul Educaţiei și Cercetării al Republicii Moldova**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică și Microelectronică**



*Laboratory work 1*

Subject: Cryptography methods for information protection

Done by: Sirețanu Andreia-Cristina

st. gr. FAF-211

Verified by: Aureliu ZGUREANU

asist. univ.

Chişinău - 2023

**Subiectul:** Cifrul lui Cesar

**Sarcini:**

1. De implementat algoritmul Cezar pentru alfabetul limbii engleze în unul din limbajele de programare. Utilizați doar codificarea literelor cum este arătat în tabelul 1 (nu se permite de folosit modificările specificate în limbajul de programare, de ex. ASCII sau Unicode). Valorile cheii vor fi cuprinse între 1 și 25 inclusiv și nu se permit alte valori. Valorile caracterelor textului sunt cuprinse între ‘A’ și ’Z’, ’a’ și ’z’ și nu sunt premise alte valori. În cazul în care utilizatorul introduce alte valori - i se va sugera diapazonul corect. Înainte de criptare textul va fi transformat în majuscule și vor fi eliminate spațiile. Utilizatorul va putea alege operația - criptare sau decriptare, va putea introduce cheia, mesajul sau criptograma și va obține respectiv criptograma sau mesajul decriptat.
2. De implementat algoritmul Cezar cu 2 chei, cu păstrarea condițiilor exprimate în Sarcina 1.1. În plus, cheia 2 trebuie să conțină doar litere ale alfabetului latin, și să aibă o lungime nu mai mică de 7.

**Cifrul lui Cesar**

Cifrul lui Cesar (sau Cezar). În acest cifru fiecare literă a textului clar este înlocuită cu o nouă literă obţinută printr-o deplasare alfabetică. Cheia secretă k, care este aceeaşi la criptare cât şi la decriptare, constă în numărul care indică deplasarea alfabetică, adică k{1, 2, 3,…, n–1}, unde n este lungimea alfabetului. Criptarea şi decriptarea mesajului cu cifrul Cezar poate fi definită de formulele c = ek(x) = x + k (mod n), m = dk(y) = y – k (mod n), unde x și y sunt reprezentarea numerică a caracterului respectiv din textul clar m și din criptograma c. Funcţia numită Modulo (a mod b) returnează restul împărţirii numărului întreg a la numărul întreg b. Această metodă de criptare este numită aşa după Iulius Cezar, care o folosea pentru a comunica cu generalii săi, folosind cheia k = 3.

**Rezultatul efectuării sarcinilor:**

public static void main(String[] args) {  
 *cesarEncryption*("Maw Maw", 2);  
 System.*out*.println();  
 *cesarDecryption*("OCYOCY", 2);  
 System.*out*.println();  
 *cesarBrutEncryption*("Maw Maw", 2, "Nya");  
 System.*out*.println();  
 *cesarBrutDecryption*("PCZPCZ", 2, "Nya");  
 System.*out*.println();  
}

Figure 1 – Main method and the scanner that interacts with user

private static void keyVerification(int key) {  
 if ((key > 25) || (key <= 0)) {  
 throw new NoSuchElementException("Introduce 0 < K < 26");  
 }  
}

Figure 2 – Function that check whether key is valid

public static void cesarEncryption(String word, int key) {  
 *keyVerification*(key);  
 List<Character> chars = new ArrayList<>();  
 for (char ch : word.toUpperCase().replaceAll(" ", "").toCharArray()) {  
 chars.add(ch);  
 }  
 List<Character> alphabetList = *getAlphabetList*();  
 alphabetList.forEach(System.*out*::print);  
 System.*out*.println();  
 chars.stream().map(character -> {  
 int i = *indexOfElement*(alphabetList, character);  
 if ((i + key) >= alphabetList.size()) {  
 return alphabetList.get(*indexOfElement*(alphabetList, character) - 26 + key);  
 }  
 return alphabetList.get(*indexOfElement*(alphabetList, character) + key);  
 }).forEach(System.*out*::print);  
}

public static void cesarDecryption(String word, int key) {  
 *keyVerification*(key);  
 List<Character> chars = new ArrayList<>();  
 for (char ch : word.toUpperCase().replaceAll(" ", "").toCharArray()) {  
 chars.add(ch);  
 }  
 List<Character> alphabetList = *getAlphabetList*();  
 alphabetList.forEach(System.*out*::print);  
 System.*out*.println();  
 chars.stream().map(character -> {  
 int i = *indexOfElement*(alphabetList, character);  
 if ((i - key) < 0) {  
 return alphabetList.get(i + 26 - key);  
 }  
 return alphabetList.get(i - key);  
 }).forEach(System.*out*::print);  
}

Figure 3 – Caesar Encryption and Decryption

public static void cesarBrutEncryption(String word, int key, String keyWord) {  
 *keyVerification*(key);  
 List<Character> chars = new ArrayList<>();  
 for (char ch : word.toUpperCase().replaceAll(" ", "").toCharArray()) {  
 chars.add(ch);  
 }  
 List<Character> alphabetList = *getAlphabetList*();  
 List<Character> keyWordList = new ArrayList<>();  
 for (char ch : keyWord.toUpperCase().replaceAll(" ", "").toCharArray()) {  
 keyWordList.add(ch);  
 }  
 List<Character> keyWordRemove = new ArrayList<>();  
 for (char ch : keyWord.toUpperCase().replaceAll(" ", "").toCharArray()) {  
 keyWordRemove.add(ch);  
 }  
 keyWordRemove.forEach(alphabetList::remove);  
 alphabetList.addAll(0, keyWordRemove.stream().distinct().toList());  
 alphabetList.forEach(System.*out*::print);  
 System.*out*.println();  
 chars.stream().map(character -> {  
 int i = *indexOfElement*(alphabetList, character);  
 if ((i + key) >= alphabetList.size()) {  
 return alphabetList.get(*indexOfElement*(alphabetList, character) - 26 + key);  
 }  
 return alphabetList.get(*indexOfElement*(alphabetList, character) + key);  
 }).forEach(System.*out*::print);  
}  
  
public static void cesarBrutDecryption(String word, int key, String keyWord) {  
 *keyVerification*(key);  
 List<Character> chars = new ArrayList<>();  
 for (char ch : word.toUpperCase().replaceAll(" ", "").toCharArray()) {  
 chars.add(ch);  
 }  
 List<Character> alphabetList = *getAlphabetList*();  
 List<Character> keyWordList = new ArrayList<>();  
 for (char ch : keyWord.toUpperCase().replaceAll(" ", "").toCharArray()) {  
 keyWordList.add(ch);  
 }  
 List<Character> keyWordRemove = new ArrayList<>();  
 for (char ch : keyWord.toUpperCase().replaceAll(" ", "").toCharArray()) {  
 keyWordRemove.add(ch);  
 }  
 keyWordRemove.forEach(alphabetList::remove);  
 alphabetList.addAll(0, keyWordRemove.stream().distinct().toList());  
 alphabetList.forEach(System.*out*::print);  
 System.*out*.println();  
 chars.stream().map(character -> {  
 int i = *indexOfElement*(alphabetList, character);  
 if ((i - key) < 0) {  
 return alphabetList.get(i + 26 - key);  
 }  
 return alphabetList.get(i - key);  
 }).forEach(System.*out*::print);  
}

Figure 4 – Caesar Brut Encryption and Decryption

public static int indexOfElement(List<Character> characterList, Character elementToFind) {  
 int index = 0;  
 for (Character element : characterList) {  
 if (element.equals(elementToFind)) {  
 return index;  
 }  
 index++;  
 }  
 return -1;  
}

Figure 5 – Function to find the index of an element in a list

private static List<Character> getAlphabetList() {  
 List<Character> alphabet = new ArrayList<>();  
 for (char letter = 'A'; letter <= 'Z'; letter++) {  
 alphabet.add(letter);  
 }  
 return alphabet;  
}

Figure 6 – Function to declare a list with alphabet characters

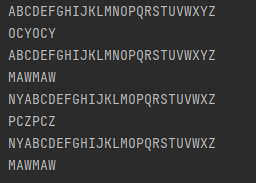


Figure 8 – Caesar Results

**Conclusion:** At the current laboratory work I studied the functionality of Caesar and Caesar Brut encryption and decryption. I created an application that encrypt and decrypt a word having the key that represents the step of the word on a list. As we can notice, the second program makes the code more secure and it is harder to understand what the sender meant because the breaker has to find 26! \* 25 versions of the word.