

**Akademia Techniczno-Humanistyczna**

**w Bielsku-Białej**

Wydział Budowy Maszyn i Informatyki

**Sprawozdanie z przedmiotu:**

**Bezpieczeństwo technologii informatycznych**

**(ćwiczenia laboratoryjne)**

**Ćwiczenie numer:**

**3**

**Temat ćwiczenia: Szyfr afiniczny**

**Wykonał: Piotr Olearczyk (nr alb. 49691)**

1. **Wykonanie programu**

**Program został napisany w języku C#**

* 1. **Opis kodu**

Na początku utworzyłem klasę statyczną **AffineCipher**, która miała za zadanie dostęp do szyfrowania oraz deszyfrowania informacji szyfrem afinicznym. Zawiera ona dwie metody – szyfrującą **Cipher** oraz deszyfrującą **Decipher.** Obie z nic przyjmują 3 parametry:

* + 1. Typu string **textToEncrypt/textToDecrypt** – tekst do szyfrowania oraz deszyfrowania
    2. Typu int **a –** liczba pierwsza mniejsza od liczby możliwych znaków kodowania **–** dla samego alfabetu jest to 26. Ja natomiast dodałem możliwość szyfrowania spacji oraz znaków interpunkcyjnych.
    3. Typu int b – liczba dowolna mniejsza od liczby możliwych znaków kodowania.

Klasa AffineCipher zawiera również inne metody prywatne: metodę walidującą, sprawdzającą największy wspólny dzielnik, która potrzebna jest do kolejnej metody sprawdzającej liczbę pierwszą.

* 1. **Klasa szyfrująca oraz deszyfrująca**

internal static class AffineCipher

{

private static readonly char[] CharCollection =

Enumerable.Range('A', 26)

.Concat(Enumerable.Range('a', 26)) // dodanie całego alfabetu

.Concat(Enumerable.Range(' ', 1)) // dodanie spacji

.Concat((new int[] { '?', '!', '.', ':', '-', '\_', '(', ')' })) // dodanie znaków interpunkcyjnych

.Select(x => (char)x)

.ToArray();

// szyfrowanie

public static string Cipher(string textToEncrypt, int a, int b)

{

if (!AssertAB(a, b)) return textToEncrypt;

var result = string.Empty;

var m = CharCollection.Length;

var result1 = result;

foreach (var pChar in textToEncrypt)

{

var p = Array.IndexOf(CharCollection, pChar);

var c = a \* p + b % m;

var cIdx = c % CharCollection.Length;

var c1 = CharCollection[cIdx];

result1 += c1;

}

return result1;

}

// Deszyfrowanie

public static string Decipher(string textToDecrypt, int a, int b)

{

if (!AssertAB(a, b)) return textToDecrypt;

var result = string.Empty;

foreach (var cChar in textToDecrypt)

{

var c = Array.IndexOf(CharCollection, cChar);

var aInverse = GetMultiplicativeInverse(a);

var pIdx = aInverse \* (c - b) % CharCollection.Length;

if (pIdx < 0)

{

pIdx += CharCollection.Length;

}

var pChar = CharCollection[pIdx];

result += pChar;

}

return result;

}

// Zwraca największy wspólny dzielnik

private static int GreatestCommonDivisor(int a, int b)

{

while (true)

{

if (b == 0) return a;

var a1 = a;

a = b;

b = a1 % b;

}

}

// Sprawdza czy są liczbami pierwszymi

private static bool AreRelativelyPrimes(int m, int n)

{

return GreatestCommonDivisor(m, n) == 1;

}

// walidacja wartości kluczy

private static bool AssertAB(int a, int b)

{

var result = false;

var message = string.Empty;

// Parametry a oraz b muszą być pomiędzy 1 <= a <= CharCollection.Length

if (a < 1 || a > CharCollection.Length)

{

message = $"'a' musi być pomiędzy [1,{CharCollection.Length}]";

}

else if (b < 1 || b > CharCollection.Length)

{

message = $"'b' musi być pomiędzy [1,{CharCollection.Length}]";

}

else if (!AreRelativelyPrimes(a, CharCollection.Length))

{

message = $"'a' musi być liczbą pierwszą do {CharCollection.Length}";

}

else

{

result = true;

}

Console.WriteLine(message);

return result;

}

// odwracanie multiplikatywne

private static int GetMultiplicativeInverse(int a)

{

var result = 1;

for (var i = 1; i <= CharCollection.Length; i++)

{

if ((a \* i) % (CharCollection.Length) == 1)

{

result = i;

}

}

return result;

}

}

* 1. **Struktura główna programu – wywołanie szyfrowania/deszyfrowania**
     1. Program został zaprojektowany w taki sposób, aby pytać użytkownika o tekst do zaszyfrowania.
     2. Następnie użytkownik musi podać klucze szyfrujące.
     3. Po wciśnięciu przycisku ENTER program zwraca zaszyfrowany tekst a następnie ten tekst odszyfrowuje drugą metodą i go zwraca.

internal class Program

{

static void Main(string[] args)

{

Console.WriteLine("Szyfr afiniczny");

Console.WriteLine();

Console.Write("Podaj tekst do zaszyfrowania: ");

var textToEncrypt = Console.ReadLine();

Console.WriteLine("Tekst do zaszyfrowania: " + textToEncrypt);

Console.WriteLine();

Console.Write("Podaj pierwszy klucz (liczba pierwsza): ");

var a = int.Parse(Console.ReadLine());

Console.WriteLine();

Console.Write("Podaj drugi klucz: ");

var b = int.Parse(Console.ReadLine());

var cipheredText = AffineCipher.Cipher(textToEncrypt, a, b);

Console.WriteLine("Zaszyfrowany tekst: {0}", cipheredText);

var decipheredText = AffineCipher.Decipher(cipheredText, a, b);

Console.WriteLine("Odszyfrowany tekst: {0}", decipheredText);

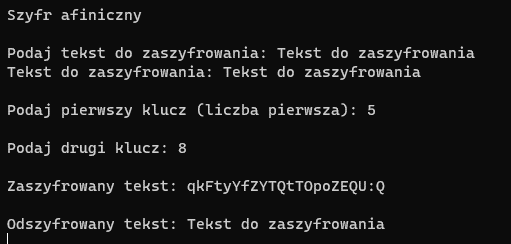
Console.ReadKey();

}

}

1. **Testy**

Testy programu w konsoli PowerShell



* 1. Jak można zobaczyć na powyższym zdjęciu ekranu celem szyfrowania był tekst „Tekst do zaszyfrowania”
  2. Klucze jakie użyłem do zaszyfrowania do 5 oraz 8.
  3. Otrzymałem zaszyfrowany tekst „qkFtyYfZYTQtTOpoZEQU:Q”
  4. Następnie program wywołał deszyfrowania zaszyfrowanego tekstu i otrzymałem „Tekst do zaszyfrowania”

1. **Wnioski**
   1. Szyfrowanie afiniczne mimo swojej skomplikowanej budowie należy do szyfrów monoalfabetycznych tzn. jednej literze alfabetu (zbioru znaków) odpowiada dokładnie jedna litera alfabetu tajnego. Znając część liter jesteśmy w stanie odszyfrować lub domyśleć się ich reszty z kontekstu.
   2. Liczba kluczy jest niewielka ponieważ wynosi 12 (liczby pierwsze) \* 26 = 312. Jest to bardzo mała liczba kluczy – wystarczy metodą prób i błędów podstawiać wartości aby uzyskać wynik.