

PROJEKT INŻYNIERSKI

„Symulator Maszyny Szyfrującej Lorenza”

Jakub KUROWSKI

Nr albumu 290575

**Kierunek:** Teleinformatyka

**PROWADZĄCY PRACĘ**

**Dr inż. Ewa Płuciennik**

**KATEDRA INFORMATYKI STOSOWANEJ**

**Wydział Automatyki, Elektroniki i Informatyki**

# GLIWICE 2022/2023

## Tytuł pracy:

Symulator Maszyny Szyfrującej Lorenza

## Streszczenie:

Prosty program symulujący działanie maszyny szyfrującej Lorenza, wykorzystywanej podczas IIWŚ do szyfrowania transmisji telegraficznych między Naczelnym Dowództwem III Rzeszy (OKW), a sztabami armii niemieckich w okupowanej Europie. Program stworzony przy pomocy języka C++, z interfejsem tekstowym, w oknie konsoli.

## Słowa kluczowe:

Program, C++, Symulator, Interfejs tekstowy

## Thesis title:

Lorenz Cipher Machine Simulator

## Abstract:

A simple program that simulates the functioning of a Lorenz cipher machine, as utilized during WWII to cipher telegraphic transmissions between the High Command of the 3rd Reich (OKW) and the various Army Commands throughout occupied Europe. The program was created with the C++ language, and functions through a text interface, in a console window.

## Keywords:

### Program, C++, Simulator, Text interface

Spis treści

[Rozdział 1 Wstęp 1](#_Toc98759118)

[Rozdział 2 Analiza tematu 3](#_Toc98759119)

[Rozdział 3 Wymagania i narzędzia 5](#_Toc98759120)

[Rozdział 4 Specyfikacja zewnętrzna programu 7](#_Toc98759121)

[Rozdział 5 Specyfikacja wewnętrzna programu 9](#_Toc98759122)

[Rozdział 6 Weryfikacja i walidacja 11](#_Toc98759123)

[Rozdział 7 Podsumowanie i wnioski 13](#_Toc98759124)

[Bibliografia 15](#_Toc98759125)

[Spis skrótów i symboli 19](#_Toc98759126)

[Źródła 20](#_Toc98759127)

[Lista dodatkowych plików, uzupełniających tekst pracy 21](#_Toc98759128)

[Spis rysunków 22](#_Toc98759129)

[Spis tablic 23](#_Toc98759130)

# Rozdział 1 Wstęp

* wprowadzenie w problem/zagadnienie
* Osadzenie problemu w dziedzinie
* Cel pracy
* Zakres pracy
* Zwięzła charakterystyka rozdziałów
* Jednoznaczne określenie wkładu autora, w przypadku prac wieloosobowych – tabela z autorstwem poszczególnych elementów pracy

Pierwsza linia akapitu z wcięciem. Czcionka Times New Roman lub Callibri 12pt. Obustronne wyrównanie. Interlinia 1.3

Rozdział 2   
  
Analiza tematu

* sformułowanie problemu
* osadzenie tematu w kontekście aktualnego stanu wiedzy (state of the art.) o poruszanym problemie
* studia literaturowe [2,3,4,1] – opis znanych rozwiązań (także opisanych naukowo, jeżeli problem jest poruszany w publikacjach naukowych), algorytmów

(1)

# Rozdział 3 Wymagania i narzędzia

* wymagania funkcjonalne i niefunkcjonalne
* przypadki użycia (diagramy UML) – dla prac, w których mają zastosowanie
* opis narzędzi, metod eksperymentalnych, metod modelowania itp.
* metodyka pracy nad projektowaniem i implementacja – dla prac, w których ma to zastosowanie

# Rozdział 4 Specyfikacja zewnętrzna programu

##### Wymagania sprzętowe i programowe

Ponieważ program działa w oknie konsoli systemu Windows, program ten nie posiada żadnych specyficznych dla siebie wymagań sprzętowych, wymagany jest tylko komputer klasy PC z zainstalowanym na nim systemem operacyjnym Windows.

###### Instalacja i aktywacja

Ze względu na to, że program działa w oknie konsoli, nie wymaga on instalacji, użytkownik musi tylko uruchomić plik wykonywalny *Jakukur.exe*.

##### Obsługa programu

Po uruchomieniu programu powinno wyświetlić się poniższe okno konsoli, na którym wyświetlone będzie menu główne programu:

**Obraz zawierający tekst

Opis wygenerowany automatycznie**

###### Rys. 4.1 – Początkowe okno programu

Ze względu na fakt, że program działa w konsoli systemu Windows, do obsługi wymagana jest klawiatura. Użycie myszki jest czysto opcjonalne, i sprowadza się do ewentualnego zaznaczania przez użytkownika tekstu wyświetlanego w oknie konsoli, na potrzeby np. skopiowania go do schowka.

Aby wybrać daną opcję podaną w menu, należy podać jej numer, poprzez wciśnięcie odpowiedniego klawisza numerycznego na klawiaturze, i potwierdzenie wyboru klawiszem *enter*. Opcje 1 – 4 tylko wyświetlają informacje dla użytkownika, zaś opcje 5 – 7 już wymagają podawania przez użytkownika danych. Opcja 8 wyłącza program. Za każdym razem, jak opcje 1 – 7 zakończą działanie, program ponownie wyświetli menu główne.

## Działania poszczególnych opcji są następujące:

1 – Wyświetla listę obsługiwanych przez program znaków, które mogą być wykorzystywane podczas kodowania, efektem aktywacji tej opcji jest poniższy tekst:

Obraz zawierający tekst, sprzęt elektroniczny, klawiatura

Opis wygenerowany automatycznie

###### Rys. 4.2 – Obsługiwane znaki tekstu kodowanego lub dekodowanego

Jak można zauważyć, program do kodowania obsługuje tylko małe litery plus liczby i wybrane znaki specjalne, podawanie przez użytkownika znaków innych niż te obsługiwane jest niezgodne z założeniami programu i spowodować jego nieprzewidzianie zachowanie.

2 – Wyświetla krótki opis oryginalnej maszyny Lorenza

3 – Wykonuje szybki test funkcjonalności kodowania i dekodowania znaków. Program wyświetla tekst przykładowy, koduje go, wyświetla tekst zakodowany, dekoduje tekst zakodowany i wyświetla tekst zdekodowany który, jeżeli wszystko powiodło się powodzeniem, powinien być identyczny do tekstu jawnego.

4 – Wyświetla obecną konfigurację kół symulowanej w programie maszyny Lorenza, od których konfiguracji zależy sposób kodowania tekstu jawnego i dekodowania tekstu zakodowanego. Jeżeli użytkownik nie skonfigurował ręcznie kół, wówczas program używa domyślnej konfiguracji kół:

Obraz zawierający tekst

Opis wygenerowany automatycznie

###### Rys. 4.3 – Wyświetlanie domyślnych sekwencji na kołach wirtualnej maszyny Lorenza

5 – Pozwala użytkownikowi na skonfigurowanie kół w programie. Po aktywacji tej opcji użytkownik **musi po kolei skonfigurować wszystkie koła wirtualnej maszyny**. Konfiguracja sprowadza się podawania dla każdego koła dowolnej sekwencji znaków (poza spacjami) o dowolnej długości, którą potem program wykorzystuje do generacji sekwencji na kołach. Efekt konfiguracji może zostać sprawdzony ponownym wywołaniem opcji 4. Na przykład, jeżeli użytkownik następująco skonfiguruje maszynę:

Obraz zawierający tekst

Opis wygenerowany automatycznie

###### Rys. 4.4 – Konfiguracja kół

Wówczas wygenerowane sekwencje prezentują się następująco:

Obraz zawierający tekst

Opis wygenerowany automatycznie

###### Rys. 4.5 – Wygenerowane sekwencje na kołach

Warto wspomnieć, że nowa konfiguracja kół jest ważna tylko na czas obecnej sesji w programie, po jego ponownym uruchomieniu, program wraca do konfiguracji domyślnej.

6 – Pozwala użytkownikowi na zakodowanie lub zdekodowanie tekstu w oknie konsoli. Użytkownik może tutaj podać tekst dowolnej długości, zawierający spacje, który zostanie w całości zakodowany przez program i wyświetlony w oknie konsoli. Alternatywnie, użytkownik może podać zakodowany ciąg znaków, i jeżeli obecna konfiguracja kół maszyny w programie jest taka sama jak w momencie tworzenia tekstu zakodowanego, to wówczas rezultatem będzie zdekodowany tekst jawny.

Przykładowo, jeżeli zakodujemy poniższy tekst:

Obraz zawierający tekst

Opis wygenerowany automatycznie

###### Rys.4.6 – Ręczne kodowanie przykładowego

To jeśli sekwencje na kołach są takie same jak były w momencie zakodowania tekstu jawnego, to wtórne podanie tekstu zakodowanego da zdekodowany tekst jawny:

Obraz zawierający tekst

Opis wygenerowany automatycznie

###### Rys. 4.7 – Ręczne dekodowanie tekstu zakodowanego w Rys. 4.6

Jeżeli zaś sekwencja jest inna, wówczas zamiast zdekodować, tekst zostanie zakodowany wtórnie:

Obraz zawierający tekst

Opis wygenerowany automatycznie

###### Rys. 4.8 – Podwójne kodowanie tekstu

Powyższe zachowanie jest zgodne z sposobem kodowania tekstu przez maszynę Lorenza, ale z punktu widzenia użytkownika jest działaniem niechcianym, dlatego ważne jest zapewnienie przez użytkownika zgodności między sekwencjami kół użytych do kodowania i dekodowania.

7 – Pozwala użytkownikowi na odczyt pliku tekstowego i jego zakodowanie (lub zdekodowanie, jeżeli plik zawiera tekst zakodowany tą samą sekwencją na kołach jak ta którą jest obecnie skonfigurowany program). Użytkownik podaje ścieżkę do pliku tekstowego wraz z jego rozszerzeniem (najlepiej *.txt*), program odczytuje plik, koduje/dekoduje jego zawartość, a następnie zapisuje wynik do nowego pliku tekstowego *output.txt*, znajdującego się w tym samym katalogu co program.

8 – Kończy pracę programu.

# Rozdział 5 Specyfikacja wewnętrzna programu

**Założenia podstawowe**

Program został napisany w języku C++ bez wykorzystywania paradygmatu programowania obiektowego. Działanie programu opiera się na pętli, w której w zależności od komendy podawanej na wejście wywoływane są odpowiednie funkcje i podfunkcje, realizujące odpowiednie elementy działania programu. Większość zmiennych wykorzystywanych w programie jest zadeklarowanych globalnie.

**Wykorzystywane biblioteki**

Program wykorzystuje tylko następujące domyślne biblioteki C++:

* ***<iostream>***, zapewniającą obsługę pobierania od użytkownika danych i wyświetlania ich przez program w oknie konsoli
* ***<string>***, pozwalającą na tworzenie i obsługę zmiennych typu *string*, dynamicznych tablic znaków
* ***<map>***, pozwalającą na tworzenie i obsługę zmiennych typu *map*, sortowanych list par klucz-wartość
* ***<fstream>***, zapewniającą obsługę odczytu z i zapisu do plików przez program
* ***<locale>***, pozwalającą na zmianę *locale* programu, co pozwala na wyświetlanie przez program polskich znaków w konsoli.

Dodatkowo, w programie jest domyślnie wykorzystywana przestrzeń nazw (*namespace*) *std*. Jednakże nie jest to funkcjonalność krytyczna do działania programu, przestrzeń została wykorzystana dla ułatwienia procesu tworzenia kodu.

**Zmienne programu**

Program wykorzystuje następujące typy zmiennych:

* Zmienne dynamicznych tablic znaków typu *string*.
* Zmienne liczb całkowitych *int*.
* Zmienne strumienia dostępu do plików *ifstream* (strumień odczytu) i *ofstream* (strumień zapisu)
* Zmienna sortowanej listy par klucz-wartość typu *map*, sortowana według kluczów komórek.

**Zmienne dynamicznych tablic znaków *string*:**

* Zmienne *chi1, chi2, chi3, chi4, chi5, mu1, mu2, psi1, psi2, psi3, psi4, psi5* służą do przechowywania sekwencji stanów na wirtualnych kołach kodowych symulowanej przez program maszyny szyfrującej Lorenza
* Zmienne *input, ouput* są wykorzystywane w procesie kodowania/dekodowania tekstu, gdzie zmienna *input* zawiera tekst do przetworzenia, który po przetworzeniu jest zapisywany do zmiennej *output*.
* Zmienna *buffer*, jak nazwa sugeruje, jest wykorzystywana jako bufor odczytu, podczas operacji na pliku tekstowym

**Zmienne liczb całkowitych *int*:**

* Zmienne *ic1, ic2, ic3, ic4, ic5, im1, im2, ip1, ip2, ip3, ip4, ip5* służą do przechowywania wartości związanych z wirtualnymi kołami kodującymi. Rodzaj tych wartości zależy od obecnego stanu maszyny – podczas kodowania tekstu, zmienne te zawierają obecne wartości 0 lub 1 znajdujących się na kołach kodujących, zaś w przypadku konfiguracji kół, zmienne te wówczas przechowują rozmiary kół, jako długość odpowiedniej zmiennej *string* reprezentującą dane koło kodujące.
* Zmienne *i, j, k* służą jako indeksy odpowiadające obecnie pobieranym wartościom z kół kodujących, *i* jest wykorzystywane dla kół chi i mu1, *j* dla kół psi, a *k* dla koła mu2.
* Zmienna *codekey* zawiera obecnie obliczoną wartość klucza kodującego
* Zmienna *encoded* zawiera wartość obecnego zakodowanego znaku
* Zmienna *control* jest zmienną kontrolną wykorzystywaną do kontroli menu w pętli głównej programu

**Zmienne strumienia dostępu do plików *ifstream* i *ofstream*.**

* Zmienna *ifstream* *infile* służy do dostępu do pliku wejściowego, z którego program pobiera dane
* Zmienna *ofstream outfile* służy do dostępu do pliku wyjściowego, do którego program zapisuje dane

**Zmienna sortowanej listy par klucz-wartość *map*.**

Zmienna *map<char,int> ITA2mod* służy do swoistej symulacji kodowania znaków ITA2 , na którym były oparte oryginalne, fizyczne maszyny szyfrujące Lorenza. W każdej komórce tej mapy znajduje się para klucz – wartość, gdzie klucz, jako typ znakowy *char*, odpowiada znakowi kodowanego standardem komputerowym ASCII, a wartość liczby całkowitej typu *int* skojarzona z danym kluczem jest wartością kodu ITA2 odpowiadającemu danemu znakowi. Warto zaznaczyć, że zmienna *ITA2mod*, nie reprezentuje kodowania ITA2 w stosunku 1:1 – jest to spowodowane zastąpieniem kodowania znaków kontrolnych dodatkowymi znakami specjalnymi. Taka realizacja została spowodowana chęcią zapewnienia większej stabilności podczas procesu odczytu i kodowania tekstu. Dodatkowo, ze względu na to że ITA2 nie rozróżnia między dużymi a małymi literami, zmienna mapy zawiera tylko małe litery, plus cyfry i wybrane znaki specjalne, oraz znak spacji – w sumie *ITA2mod* przechowuje 64 różnych par.

**Opis funkcji**

Funkcjonalnie program podzielony jest na dwa typy funkcji, *główne* i *pomocnicze*.

Funkcje główne są bezpośrednio wywoływane w pętli głównej programu poprzez instrukcję *switch* i poza trzema wyjątkami zawierają wywołania funkcji pomocniczych i dodatkowe, proste działania, takie jak wyświetlanie tekstu czy też czyszczenie zmiennych. Celem tych funkcji jest kontrola wykonywania danej funkcjonalności programu, która jest realizowana przez odpowiednie wywoływanie funkcji pomocniczych.

Funkcje pomocnicze służą do wykonywania powtarzalnych działań przez funkcje główne, ich istnienie pozwala na lepszą organizację kodu podczas jego tworzenia – dla przykładu, jakiekolwiek problemy z daną funkcjonalnością programu łatwiej naprawić modyfikując funkcje pomocnicze wywoływane przez funkcję główną, niż modyfikować jedną, bardzo rozwiniętą funkcję. Jak wcześniej wspomniano, są wywoływane przez funkcje główne, w niektórych przypadkach więcej niż raz.

**Pętla główna programu**

Pętla główna programu jest zawarta w funkcji *int main()*, której większość należy do pętli *while*. Przebieg działania jest następujący:

* *locale* programu jest ustawiany na obsługę polskich znaków i formatów dat i liczb, poprzez funkcję biblioteczną *setlocale(LC\_ALL, „pl\_PL”).*
* Program wyświetla tekst *Symulator maszyny Lorenza*
* Program wchodzi w pętlę *while*. Pętla ta będzie działać w nieskończoność dopóki wartość globalnej zmiennej kontrolnej *int control* jest różna od 8 (domyślnie, wartość zmiennej wynosi 0) .Przebieg pętli *while* jest następujący:
  + Wykonywana jest instrukcja *switch*, która w zależności od obecnej wartości zmiennej *control* wywołuje odpowiednie funkcje. Reaguje on na wartości *control* będącymi liczbami całkowitymi od 1 do 7. Przy pierwszej iteracji pętli, która odbywa się po uruchomieniu programu, *control* jest równy wartości domyślnej 0, dlatego *switch* nie wykona się.
  + Wywoływana jest funkcja *void attract()*, która wyświetla w oknie konsoli tekst menu.
  + Program czeka na podanie przez użytkownika wartości zmiennej *control*. Podanie wartości liczby całkowitej z zakresu od 1 do 7 sprawi, że następna iteracja pętli wywoła odpowiedni *case* instrukcji *switch*. Podanie wartości 8 powoduje spełnienie warunku wykonywania pętli *while*, a co za tym idzie zakończy zarówno pętlę, jak i działanie programu, gdyż pętla *while* jest ostatnim elementem wykonywanym w *int main()*. Podanie innych wartości całkowitych spowoduje ponowne wykonanie się pętli *while* tak jak w przypadku pierwotnego uruchomienia programu – pojawia się ponownie menu. Jeżeli jednak użytkownik poda wartość niezgodną z typem zmiennych *int* jaką jest zmienna *control* – np. literę, znak specjalny, ciąg znaków, czy też liczbę niecałkowitą – wówczas następuje działanie odwrotne od zamierzonego i program zapętla się, i wymagany jest wówczas jego restart.

**Funkcje główne**

Funkcje główne są podawane zgodnie z kolejnością z jaką są osadzone w pętli głównej programu, wraz z podaniem wywoływanych przez nie funkcji pomocniczych, a także wartości *control* wywołujących je.

* **void texthelp(map<char, int>input)**
  + Wywoływana dla *control* równego **1**.
  + Nie wywołuje żadnych funkcji pomocniczych
  + Jedna z trzech funkcji głównych, które nie wywołują funkcji pomocniczych, i które można byłoby skategoryzować za pomocnicze, gdyby nie fakt, że są bezpośrednio wywoływane przez użytkownika poprzez instrukcję *switch* w pętli głównej programu.
  + Funkcja ta ma za zadanie wyświetlenie obsługiwanych przez program znaków, które mogą być poprawnie zakodowane i zdekodowane, innymi słowy, wyświetla ona zawartość zmiennej *map<char,int> ITA2mod*. Funkcja iteruje kolejno po komórkach mapy i wyświetla skojarzoną z daną komórką wartość klucza. Dla potrzeb estetycznych, znaki są wyświetlane w liczbie 14 znaków na linię
* **void lorenzinfo()**
  + Wywoływana dla *control* równego **2**.
  + Nie wywołuje żadnych funkcji pomocniczych
  + Kolejna funkcja, którą można byłoby skategoryzować za pomocniczą. Służy tylko i wyłącznie do wyświetlania krótkiej informacji tekstowej na temat historii i działania maszyny szyfrującej Lorenza.
* **void test()**
  + Wywoływana dla *control* równego **3**.
  + Wywołuje funkcję pomocniczą *void* *encode(string toencode)*.
  + Pierwsza funkcja główna wywołująca funkcje pomocnicze, *void test()* jak nazwa sugeruje, służy do szybkiego testu funkcjonalności symulatora przy obecnej konfiguracji kół kodujących. Funkcja zapisuje do zmiennej *string input* prosty tekst, który jest następnie przekazywany do podfunkcji *encode*, która koduje podany ciąg znaków, który jest zapisywany w zmiennej *string output*. Następnie, *test* wyświetla w oknie konsoli oryginalny tekst wraz z jego wersją po zakodowaniu. Kolejnym krokiem jest tekst dekodowania – do zmiennej *input* przypisywana jest zawartość zmiennej *output*, która jest zaś czyszczona z obecnej zawartości przy pomocy funkcji bibliotecznej *.clear()* właściwej dla biblioteki *<string>*. Następnie znowu *input* jest przekazywany do *encode*, i jeżeli wszystko powiodło się pomyślnie, zwrócona nam nowa zawartość *output*, która zostanie wyświetlona w oknie konsoli powinna być równoważna z oryginalnym tekstem testowym. Na sam koniec działania funkcji *test*, zmienne *input* i *output* są opróżniane z zawartości, znów przy pomocy zmiennej bibilotecznej *.clear()*.
* **void wheelinfo()**
  + Wywoływana dla *control* równego **4**.
  + Nie wywołuje żadnych funkcji pomocniczych.
  + Ostatnia z funkcji quasi-pomocniczych, *wheelinfo()* wyświetla obecne sekwencje na wirtualnych kołach kodujących symulowanej maszyny szyfrującej Lorenza. Ze względu na fakt, że sekwencje na kołach są przechowywane w programie jako zmienne *string*, działanie tej funkcji sprowadza się do wyświetlania tychże zmiennych.
* **void configwheels()**
* **void writeandcode()**
* **void filecode()**

**Funkcje pomocnicze**

# Rozdział 6 Weryfikacja i walidacja

* sposób testowania w ramach pracy (np. odniesienie do modelu V)
* organizacja eksperymentów
* przypadki testowe, zakres testowania (pełny/niepełny)
* wykryte i usunięte błędy
* opcjonalnie wyniki badań eksperymentalnych

|  |  |
| --- | --- |
| Tabela 4.2. Nagłówek tabeli jest nad tabelą. | |
| Poziom 1 | 24 pt |
| Poziom 2 | 20 pt |
| Poziom 3 | 16 pt |
|  | |

# Rozdział 7 Podsumowanie i wnioski

* uzyskane wyniki w świecie postawionych celów i zdefiniowanych wyżej wymagań
* kierunki ewentualnych danych prac (rozbudowa funkcjonalna …)
* problemy napotkane w trakcie pracy

# Bibliografia

1. Imię Nazwisko, Imię Nazwisko. *Tytuł książki*. Wydawnictwo, Warszawa, 2017.
2. Imię Nazwisko, Imię Nazwisko. Tytuł artykułu w czasopiśmie. *Tytuł czasopisma*, 157(8):1092–1113, 2016.
3. Imię Nazwisko, Imię Nazwisko, Imię Nazwisko. Tytuł artykułu konferencyjnego. *Nazwa konferencji*, str. 5346–5349, 2006.
4. Autor, jeśli znany. https: [www.adres.strony](http://www.adres.strony) (dostęp:dzień.miesiąc.rok)

Dodatki

# Spis skrótów i symboli

*OKW* Naczelne Dowództwo Sił Zbrojnych (niem. *Oberkommando das Wehrmacht)*

# Źródła

Jeżeli w pracy konieczne jest umieszczenie długich fragmentów kodu źródłowego,

należy je przenieść do tego miejsca.

# Lista dodatkowych plików, uzupełniających tekst pracy

W systemie, do pracy dołączono dodatkowe pliki zawierające:

* źródła programu,
* dane testowe
* film pokazujący działanie opracowanego oprogramowania lub zaprojektowanego i wykonanego urządzenia,
* itp.

# Spis rysunków

4.1 Podpis rysunku jest pod rysunkiem 12

5.1 Pseudokod w listings 14

5.2 Pseudokod w minted 14

# Spis tablic

6.1 Opis tabeli nad nią 16