Warszawa, 26.01.2016  
Politechnika Warszawska  
Wydział Elektroniki i Technik Informacyjnych

**Projekt INTEL**

**Mini Enigma**

**(3.5)**

**Architektura komputerów**

**prowadzący: mgr inż. Zbigniew Szymański**

**Tatsiana Lukashevich**

**Spis treści:**

**1.**Opis struktury programu………………..**3**

**2.**Opis struktur danych i ich implementacji, implementacji algorytmu………………….**6**

**3.**Opis testów……………………………....**8**

***1. Opis struktury programu***

Program powinien zaszyfrować tekst i zdeszyfrować tekst o odpowiednio podanych ustawieniach wirników. Po zaszyfrowaniu/zdeszyfrowaniu jednego znaku jeden z wirników obraca się o jedną pozycję.  
Tekst do zaszyfrowania/zdeszyfrowania zapisuje się w pliku o nazwie *plaintext.txt.* Ustawienie wirników zmienia się w pliku *init.txt*. Zaszyfrowany/zdeszyfrowany tekst zapisuje się do pliku *ciphertext.txt.* Jak połączone wirniki są zapisane w pliku *rotors.txt*.

Wymagania dotyczące pliku *plaintext.txt*:

-tekst jawny zawiera znaki o kodach ASCII z przedziału <32dec, 95dec> wszystkie znaki, których kody ASCII nie mieszczą się w przedziale wymienionym w poprzednim punkcie są ignorowane

- plik będzie wczytywany wiersz po wierszu (a nie na jeden raz w całości) po wczytaniu kolejnego wiersza algorytm szyfrujący kontynuuje pracę (nie powraca do stanu początkowego)

-dane kończą się pustym wierszem zawierającym tylko znak o kodzie 10dec

Plik o nazwie *rotors.txt* zawiera opis połączeń bębnów szyfrujących:

*Przykład:*

Rotorjeden;nr styku wejscia - nr styku prawa strona

00-04

01-10

02-12

….

Powyższe dane oznaczają, że np. styk wejścia 00 jest połączony z prawym stykiem wirnika pierwszego 04.

Każda sekcja składa się z etykiety (dowolny ciąg znakowy) oraz 52 linii opisujących połączenia styków po prawej i po lewej stronie bębna (dla bębnów standardowych) i 26 linie dla bębna odwracającego(alfabet łaciński). Połączenia bębnów można odnaleźć na stronie http://habrahabr.ru/post/217331/

Początkowe położenie bębnów szyfrujących podane jest w pliku *init.txt*:

*Przykład:*

22

05

12

Dane z powyższego przykładu oznaczają, że:

styk 22 pierwszego bębna sąsiaduje ze stykiem wej./wyj. o numerze 00,

styk 05 drugiego bębna sąsiaduje ze stykiem 22 pierwszego bębna i stykiem 12 trzeciego bębna

styk 12 trzeciego bębna sąsiaduje ze stykiem 05 drugiego bębna i stykiem 00 bębna odwracającego.

Zaszyfrowane ciągi znakowe powinny być umieszczone w kolejnych wierszach pliku o nazwie *ciphertext.txt*.

W programie została zaimlementowana w asemblerze funkcja dokonująca szyfrowanie zadanym alorytmem jednego wiersza tekstu. Prototyp funkcji w języku C:

void kodowanie(char \*buf1, char \*buf2, ustawienia \*ust);

Parametry:

char \*buf1 - wskazanie na bufor zawierający tekst wejściowy(do zaszyfrowania bądź odszyfrowania)

char \*buf2 - wskazanie na bufor zawierający tekst wyjściowy (zaszyfrowany bądź odszyfrowany)

ustawienia \*ust - struktura służąca do przekazania dodatkowych danych potrzebnych do prawidłowego działania algorytmu

Odczyt danych z pliku i Alokacja pamięci na bufory jest realizowany na poziomie języka C

***2.Opis struktur danych i ich implementacji, implementacji algorytmu***

Język C:

**int s1,s2,s3;**  Przechowanie ustawień wirników

**char buf1[1025];** Przechowywanie symboli wejściowych

**char buf2[1025];** Przechowywanie symboli wyjściowych

**char tablica[100];** Tablica pomocnicza

**int tablica2[2000];** Przechowywanie pierwszej części ustawień rotorów

**int tablica3[400];**  Przechowywanie drugiej części ustawień rotorów

**void read\_init();** Funkcja dla odczytu ustawień wirników

**void read\_rotors();**  Funkcja dla odczytu połączeń wirników

**int read(FILE \*file);**  Funkcja dla odczytu z pliku wirników, zmieniając na liczby

**struct ustawienia**  Struktura dla poprawnego kodowania/zdekodowania

**{**

**int \*bufor2;** Wskaźnik na tablicę 1-j części znaków do zakodowania

**int \*bufor3;** Wskaźnik na tablicę 2-j części znaków do zakodowania

**int t4;** Ustawienie początkowe 1-go wirnika

**int t5;** Ustawienie początkowe 2-go wirnika

**int t6;** Ustawienie początkowe 3-go wirnika

**};**

Intelx86:

**edi** - przechowuje wskaźnik na ciąg wejściowy

**ecx**- przechowuje wskaźnik na ciąg wejściowy

**eax** - przechowuje wskaźnik na strukturę

**ebx** - przechowuje wskażnik na bufory

**[ebp + 8]** - przechowuje pierwszy argument funkcji kodowanie

**[ebp + 12]** - przechowuje drugi argument funkcji kodowanie

**[ebp + 16]** - przechowuje trzeci argument funkcji kodowanie

**[ebp - 4]** - przechowuje pierwszy obrót wirnika

**[ebp - 8]** - przechowuje drugi obrót wirnika

**[ebp - 12]** - przechowuje trzeci obrót wirnika

**[ebp - 16]** - przechowuje miejsce w buforze dla zakodowania

**[ebp - 20]** - przechowuje znak zakodowany/zdekodowany

**bl** - zmienna pomocnicza

**al** - zmienna pomocnicza, przechowująca numer znaku w buforze

**kodowanie:** Służy do kodowania/zdekodowania znaków

**przechodz:** Przechodzi po symbolom wejściowym i szyfruje/odszyfruje je

**przepusc:** Przepuszcza znaki, które nie są dużymi literami

**zapisz:**  Zapisuje zakodowany/zdekodowany znak

**koniec:** Zapisuje wszystkie wirniki obrócone do tej pory

**rotate:** Obraca wirniki po zakodowanym/zdekodowanym znaku

**countt4:** Obraca o jedną pozycję pierwszy wirnik

**countt5:** Obraca o jedną pozycję drugi wirnik

**countt6:** Obraca o jedną pozycję trzeci wirnik

***3.Opis testów***

Testy zostały prowadzony za pomocą wpisania do pliku plaintext.txt tekstu do zaszyfrowania, wynik został zapisany do pliku ciphertext.txt. Zatem do sprawdzenia zakodowanego tekstu umieszczam go w plik plaintext.txt, porównując wyniki, w pliku ciphertext.txt i tekstu jawnego, możemy decydować o poprawności zaszyfrowanych danych za pomocą programu.  
Poniższa tabela przedstawia wyniki testowania:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tekst do zaszyfrowania | Zaszyfrowany  tekst | Zdeszyfrowany tekst | Ustawienie wirników | Komentarz |
| UPGR  HNQ  M  OL | QTXH  RBU  A  LO | UPGR  HNQ  M  OL | 00  24  12 | Jest dobrze |
| GHMOLSNUTG  BACYLPD  WQGFNZJYSER  XI | KYFUTQJOLK  CXBHTVE  RSKMJINHQDW  AZ | GHMOLSNUTG  BACYLPD  WQGFNZJYSER  XI | 00  17  20 | Jest dobrze |
| HDKTENCLOSJD  J  MQZ  P  PRY  ASW  B  GFDjd  NsK  H | XVUQJCNIYAEV  E  RTP  Z  ZMO  SAF  G  BWV  CU  X | HDKTENCLOSJD  J  MQZ  P  PRY  ASW  B  GFD  NK  H | 00  10  18 | Jest dobrze  Pomijamy przy kodowaniu małe litery,bo nie wchodzą w obszar podany w wymaganiach(32dec, 95dec) |
| JSUEHFNSI  AB  OPZQ  M | EFYJMSAFW  NX  QRDO  H | JSUEHFNSI  AB  OPZQ  M | 00  10  11 | Jest dobrze |
| ESLONVGFJDM  PDM4SC  YRN  A | OCXEUWFGAPT  DPT4CS  HBU  J | ESLONVGFJDM  PDM4SC  YRN  A | 00  01  03 | Jest dobrze  Dopuszczam przechodzenie znaku bez kodowania(32dec-65dec;90dec-95dec) |