Felhasználói dokumentáció

A projekt célja egy Enigma gép implementálása. Az enigma gépet a 2. világháborúban használták a németek titkos üzenetek kódolására és dekódolására. Lényegében egy írógép ami, ha lenyomsz egy betűt, az előtte lévő abc panelen felvillant egy másik betűt ami a kódolt párja lesz. Minden betű lenyomása után fordulnak egyet a gépben lévő tárcsák és a betű kódok megváltoznak.

Az üzenet kódolásához szükség van egy alap tárcsa beállításra (ez lesz a titkosítás kulcsa) majd a gépen egymás után be kell pötyögni a szöveget lejegyezve a felvillanó betűket. Minden betű kód párban szerepel. Ha egy lépésben az A betűből R lesz akkor ugyanabban a lépésben az R betűből A. (A->R, R->A). Így a kódolás és dekódolás folyamata gyakorlatilag ugyanaz: beállítani a korábban egyeztetett kulcsra a gépet és beírni a kódolni vagy dekódolni kívánt üzenetet.

Az eredeti gépben 5 tárcsából kell 3-at valamilyen sorrendben behelyezni a gépbe. Beállítani mind a 3 tárcsát az angol abc 26+1 betűjének egyikére (+1 a space karakter). Továbbá lehetőség van mégegyszer manuálisan 10 darab betűpárt beállítani ezzel növelve mégjobban a lehetséges kulcsok számát.

Ebből a projektben csak 3 tárcsát, előre meghatározott sorrendben és az ezeken létrehozható 27*27*27 lehetséges kulcsot fogom implementálni.

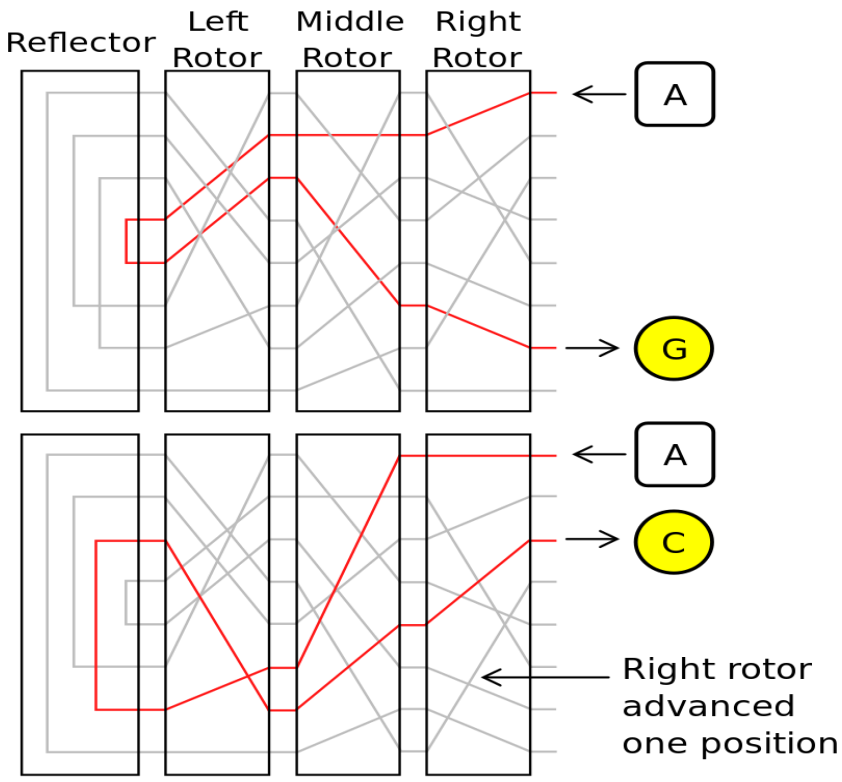
A program kéri a user-től a a kódolni kívánt üzenetet tartalmazó txt fájl nevét valamit annak a txt fájlnak a nevét amibe a kódolt üzenetet kiírjuk. Ezután a user megadja az egyes tárcsák állását.

A program lefut: feldolgozza az üzenetet és kiírja a kódot.

Programozói dokumentáció

A gépben összesen 4 tárcsa van. Ebből 3 forog és az utolső az un. Reflektor tárcsa nem.

A betűjel először végig megy a 3 tárcsán majd a reflektoron és onnan visszafelé a 3 tárcsán keresztül.



Egy tárcsa implementációjához először szükségünk lesz egy betűk összeköttetését tároló adatszerkezetre.

Mivel ez dinamikusan változik ezért célszerű pointereket használni. A betűket abc sorrendben egy láncolt listába tesszük, majd kódjuknak megfelelően párokba rendezzük. Egy betűpárt oda-vissza össze kell kötni mert használni fogjuk a kódolás függvényét és annak inverzét is.

Egy betű:

```
typedef struct Betu {
    char betu;
    struct Betu *next;
    struct Betu *pair;
} Betu;
```

Egy tárcsa a kezdő betűt és a tárcsa állását tartalmazza:

```
typedef struct {
    int allas;
```

```
    Betu *pre_head;
    Betu *post_head;
} Tarcsa;
```

pre_head: a tárcsa reflektortól távolabb eső végének a betűlistája

post_head: a tárcsa reflektorhoz közelebb eső végének a betűlistája

A tárcsák fizikai állapotát (kezdeti betűkeverés) hardcode-oljuk:

```
char alphabet[27], tarcsa1_order[27], tarcsa2_order[27],
tarcsa3_order[27], reflektor_order[27]
```

Az alphabet tömb az angol abc betűit sorrendben, + a space karaktert a végén tartalmazza.

A tárcsa orderek az abc egyadott tárcsához tartozó keverését tartalmazzák.

Pl.: ha alphabet[3] == 'c' és order[3] == 'f' akkor c párja f lesz az adott tárcsában.

Szükség lesz a többi függvényhez egy Betu megtaláló függvényre ami megkeresi egy betű karakter alapján a Betű listában a használni kívánt Betűt. A reverse megmondja ,hogy a tárcsán belül melyik betűlistánál vagyunk. Lehet 0 vagy 1 értékű. Ha 0 akkor a reflektor felé ha 1 akkor az onnan el felé néző végében lévő Betű listában kereünk.

```
Betu *search_betu(char betu, Tarcsa *tarcsa, int reverse);
```

A tárcsákat tudnunk kell forgatni. Az egyes tárcsák forgatásához a függvény:

```
void *update_tarcsa(Tarcsa *tarcsa);
```

A függvény a head-től kezdve egyesével végig megy a betűkön a next-et használva és újrahuzalozza a párokat. Utoljára “elforgatja a tárcsát”: növeli az állását 1-el és a 26-os maradékát veszi.

Ha teljesen körbefordult egy tárcsa akkor az utána következőt el kell fordítani 1-el. Szükségünk lesz egy update_osszes függvényre ami ezt megteszi:

```
void *update_osszes(Tarcsa *tarcsa_1, Tarcsa *tarcsa_2, Tarcsa *tarcsa_3);
```

A függvény forgat egyet a tarcsa1-en majd ha érzékeli ,hogy az egyik tárcsa állása 0 (teljesen körbeért) akkor updateli a következőt is.

A tárcsát a user által megadott kezdő állásba forgató függvény:

```
void kezdo_allas_tarcsa(Tarcsa *tarcsa, int kezdo_allas);
```

A függvény “kezdo_allas”-szor hívja az update_tarcsa függggvényt (a reflektornál ez mindig 0).

Szükség van egy tárcsákat inicializáló függvényre is. Ez létrehozza a tárcsát a kezdeti huzalozással együtt (alphabet->order) majd beforgatja a user által megadott állásba (meghívja a kezdo_allas_tarcsa() függvényt).

```
Tarcsa *init_tarcsa(char alphabet[27], char order[27], int kezdo_allas);
```

Az egymás után érkező betűk kódolásához a

```
char get_kod(char betu, Tarcsa *tarcsa1, Tarcsa *tarcsa2, Tarcsa *tarcsa3,
Tarcsa *reflektor);
```

függvényt használjuk.

Ez végigfuttatja a betűt a tárcsákon a reflektorig és onnan visszafelé. Utána meghívja az update_osszes() függvényt. Végül pedig visszatér a kódolt betűvel.

Utoljára a főprogramban pedig beolvassuk a user által megadott fájlból egyesével a betűket és kiírjuk őket a a szintén user által megadott eredmény fájlba.

Teszt dokumentáció

A programot először a code_test.txt -vel teszteltem. A program sikeresen és gyorsan lefutott a karakterek száma soronként ugyanannyi volt tehát minden karakterhet 1 db kódolt pár tartozott.

Két különböző sorba ugyanaz volt írva és ezeknek a kódolása különböző volt. Tehát működött a tárcsák forgatása. A programot különböző tárcsa állásokkal is futtattam, ilyenkor ennek megfelelően különböző kódolásokat kaptam.

Ezek után visszafelé futtattam a decode_test.txt -vel ami a code_test.txt -nek a kódolását tartalmazza. Az eredeti üzenetet a program sikeresen kiírja.

Végül teszteltem a programot az error_test.txt -n amiben a program által nem támogatott karakterek is voltak. Ilyenkor sikeresen ki lett írva a hibaüzenet és a program 1 visszatérítási értékkel leállt.