
Systeemprogrammeren - Oefeningenset 2
Eendimensionale rijen

Contact

sysprog@lists.ugent.be

Verantwoordelijke lesgevers

prof. dr. ir. Filip De Turck

prof. dr. Bruno Volckaert

1 Inleiding: de ADFGX-codering

In deze oefening zal de ADFGX-codering geïmplementeerd worden. Het Duitse leger ontwikkelde tussen 1914 en 1918 talloze codes en encryptietechnieken, die veelal erg moeilijk te ontcijferen waren. Omdat het onderscheppen van vitale informatie onmisbaar was voor de geallieerde troepen, telde Frankrijk tijdens de oorlog maar liefst vijf onderzoeksgroepen die zich concentreerden op het ontsleutelen van Duitse berichten. Het "Bureau du Chiffre" was reeds in 1870 opgericht, nadat Frankrijk de Frans-Duitse oorlog verloren had die zou leiden tot het oprichten van het verenigde Duitse Keizerrijk. Een van de meest gekende onderzoekers aan dit instituut was Lt. Georges Painvin, een paleontoloog en geoloog die zich pas bij het uitbreken van de oorlog begon te interesseren voor encryptietechnieken. Hij ontcijferde talloze codes, maar het kraken van de ADFGX-codering wordt aanzien als zijn grootste prestatie.



Figuur 1: Een (onopgelost) ADFGX-versleuteld bericht in Call of Duty: Black Ops (Mob of the Dead).

De ADFGX-codering werd in maart 1918 ontwikkeld door Kol. Fritz Nebel, die verantwoordelijk was voor de communicatie binnen het Duitse leger. In een eerste stap worden de ondersteunde karakters in willekeurige volgorde in een tabel geplaatst, bijvoorbeeld zoals hieronder weergegeven voor de letters in ons alfabet (waarbij de letter J gelijkgesteld wordt aan de letter I):

	A	D	F	G	X
A	Q	I	K	W	A
D	H	U	Y	X	T
F	N	M	Z	G	C
G	R	D	B	L	P
X	F	O	V	S	E

Op deze manier wordt het karakter 'a' vertaald door "AX", 'b' door "GF" en de zin "wapens en munitie geleverd" door "AGAXGXXXFAXGXXAFFDDFAADDXADXXFGXXGGXXXFXGAGD", waarbij spaties in de zin verwijderd werden. De reden dat de letters 'A', 'D', 'F', 'G' en 'X' gebruikt werden is omdat deze in Morsecode erg van elkaar verschillen, wat de kans op transmissiefouten aanzienlijk verkleinde. Merk op dat de symbolen niet vastliggen: later werd ook gebruik gemaakt van 'A', 'D', 'F', 'G', 'V' en 'X' met 36 ondersteunde karakters, en ook andere combinaties zijn mogelijk. Het resultaat van deze eerste stap resulteert in een Polybiuscodering¹, die relatief eenvoudig te ontcijferen is. Daarom worden de letters in een tweede stap verder door elkaar gehaald, door gebruik te maken van een transpositiesleutel. Beschouw bijvoorbeeld het woord "computer", waaronder de gecodeerde boodschap ("wapens en munitie geleverd") rij per rij ingevoegd wordt:

¹https://en.wikipedia.org/wiki/Polybius_square

C	O	M	P	U	T	E	R
A	G	A	X	G	X	X	X
F	A	X	G	X	X	A	F
F	D	D	D	F	A	A	D
D	X	A	D	X	X	F	G
X	X	G	G	X	X	X	F
X	X	G	A	G	D		

De kolommen worden vervolgens verplaatst, zodat de letters van de transpositiesleutel in alfabetische volgorde geplaatst worden:

C	E	M	O	P	R	T	U
A	X	A	G	X	X	X	G
F	A	X	A	G	F	X	X
F	A	D	D	D	D	A	F
D	F	A	X	D	G	X	X
X	X	G	X	G	F	X	X
X		G	X	A		D	G

Indien hetzelfde karakter meermaals voorkomt in de transpositiesleutel, wordt de oorspronkelijke volgorde van de letters gerespecteerd bij het rangschikken van de verschillende kolommen. In een laatste stap worden alle letters kolom na kolom uitgeschreven, zodat als oplossing "AFFDXXXAAFXAXDAGGGADXXXXGDDGAXFDGFXAXXDGXFXXG" verkregen wordt.

Om dit woord te decoderen, moet de transpositiesleutel gekend zijn. Op die manier kunnen er in het gecodeerde bericht twee spaties op de juiste plaatsen ingevoegd worden (initieel geïntroduceerd door de lege karakters in de kolommen 'E' en 'R') en kan de hierboven getoonde tabel terug opgesteld worden. Vervolgens worden de kolommen gerangschikt volgens de letters in de transpositiesleutel en kunnen alle karakters twee per twee vertaald worden aan de hand van de decoderingstabel uit de eerste stap.

Het Duitse leger veranderde de gekozen transpositiesleutel dagelijks en ging er vanuit dat de code niet te kraken was. Na drie slapeloze maanden slaagde Painvin er in juni 1918 toch in de code te ontcijferen, op basis van een uitgebreide analyse van onderschepte berichten.

2 Opgave

In deze oefening zal bovenstaande encryptietechniek geïmplementeerd worden met behulp van eendimensionale structuren in C. Zo zullen we onder meer bestanden inlezen en ernaar wegschrijven, karakters een voor een gaan encoderen en de resulterende matrix transponeren om het gewenste resultaat te bekomen. Neem voor je begint zeker de twee appendices met codefragmenten even door (en voer ze desgewenst even uit), zodat je vertrouwd kunt geraken met een aantal best practices. Eens je daarmee klaar bent, kan je aan de slag gaan met de volgende beschikbaar gestelde bestanden:

- `main.c`: De `main()`-functie wordt in dit bestand volledig meegeleverd. Deze zal de te implementeren functies oproepen. Dit bestand mag naar believen aangepast worden bij het testen van de code, maar bij de evaluatie van de geïmplementeerde code zal het originele bestand gebruikt worden;
- `adfgx.h`: Het headerbestand voor de encoder/decoder. Dit bestand bevat de definitie van een struct die gebruikt zal worden bij het encoderen en decoderen van tekst. Daarnaast bevat het ook de signatuur van de te implementeren functies. Dit bestand mag niet aangepast worden;
- `adfgx.c`: Dit bestand bevat de te implementeren functies, die hieronder omschreven worden;
- `settings.txt`: Een invoerbestand dat de ADFGX-codering beschrijft, op basis van vier strings die het ondersteunde alfabet, de gebruikte symbolen bij encryptie, een woordenboek en de transpositiesleutel voorstellen;
- `text.txt`: Een invoerbestand dat de te encrypteren tekst bevat;
- `encoded.txt`: Een invoerbestand dat de met brute-force te decrypteren tekst bevat;
- `permutations.txt`: Een invoerbestand voor brute-force decoding.

2.1 Hulpfuncties

Allereerst worden er een aantal hulpfuncties geïmplementeerd in `adfgx.c`:

- `void validation_check(void* input, char* description)`: Deze functie controleert of het eerste argument verschillend is van NULL. Indien dit niet het geval is, wordt een foutboodschap weergegeven die meer duiding geeft bij het object waarvan sprake (bijvoorbeeld "file pointer naar encoded.txt"). Vervolgens wordt `getchar` gebruikt om bovenstaande foutmelding leesbaar te maken, waarna het uitvoeren van het programma beëindigd wordt;
- `char* allocate_strlen(unsigned long strlen)`: Deze functie allocceert geheugen voor een karakterrij van lengte `strlen + 1` (het EOS-karakter). Aangezien we de lengte op voorhand kennen, en we ervan uitgaan dat de rij volledig opgevuld zal worden, gebruiken we `malloc` (waarom niet `calloc`?). Controleer of de allocatie succesvol was met behulp van `validation_check`, voeg het EOS-karakter in op de juiste plaats en geef de karakterrij terug;
- `char* deep_copy(const char* s)`: Neemt een diepe kopie van de inputstring, door gebruik te maken van de functie `strcpy`. Zorg ervoor dat voldoende geheugen vrijgemaakt wordt, en dat er gecontroleerd wordt of de allocatie wel succesvol was (gebruik met andere woorden de functie `allocate_strlen`);

- `char* sort(const char* s)`: Geeft een string terug die de gesorteerde versie van de inputstring bevat. De output van `sort("wagen")` is dus "aegnw";
- `int get_index(const char c, const char* s)`: Geeft de index van een karakter c in een string s terug, en -1 als het karakter niet voorkomt. Wanneer een karakter meermaals optreedt, wordt de kleinste index teruggegeven. De output van `get_index('a', "bagage")` is dus 1.
- `char* read_from_file(const char* filename)`: Met deze functie kan een bestand, geïdentificeerd door de parameter filename, ingelezen worden in een karakterrij. Hiertoe worden de functies `fseek`, `ftell` en `fread` gebruikt (merk op dat het resultaat van deze twee functies verschillend kan zijn in Windows, gezien de carriage return bij een nieuwe lijn, en dat er in dit geval een reallocatie nodig zal zijn). Voor het inlezen van tekstbestanden in C, kan een voorbeeld teruggevonden worden in Appendix A achteraan deze opgave (let op: hier wordt een vast aantal karakters gebruikt). Indien er problemen optreden bij het inladen van de tekstbestanden (controleer in de code steeds of het gegeven bestand wel geopend kan worden), kijk dan eerst of deze in dezelfde map als het project staan;
- `char* filter(const char* alphabet, char* text)`: Past de inputstring text aan, zodat enkel de (upercase) variant van de letters in alphabet voorkomen. Het resultaat van `filter("ABCDEFGHIKLM NOPQRSTUVWXYZ", "WAPENS en MUNITIE geleverd")` is bijvoorbeeld "WAPENSENMUNITIEGELEVERD". Merk op dat er een reallocatie nodig is wanneer er karakters verwijderd worden uit de string. Merk op de pointer naar de string teruggegeven moet worden (waarom?);
- `char* transpose(const char* matrix, unsigned int width)`: Transponeert een eendimensionale representatie van een matrix. Hierbij wordt de matrix rijgewijs voorgesteld door de string matrix, en stelt width het aantal kolommen voor. Er hoeft niet gecontroleerd te worden of de grootte van de matrix wel een veelvoud is van de opgegeven breedte. De output van `transpose("ABCDEF", 3)` is gelijk aan "ADBEFC", aangezien:

$$\begin{pmatrix} A & B & C \\ D & E & F \end{pmatrix}^T = \begin{pmatrix} A & D \\ B & E \\ C & F \end{pmatrix}$$

Merk op dat er in deze functie wel geheugen gealloceerd dient te worden, wat in `filter` niet het geval was.

2.2 Encodergerelateerde functies

Naast de hulpfuncties onderscheiden we ook een aantal encodergerelateerde functies. Deze vereisen naast een basiskennis van de programmeertaal C, ook een inzicht in de onderliggende algoritmen. In het opgavebestand werd dan ook de nodige pseudo-code voorzien. De volgende functies dienen geïmplementeerd te worden:

- `adfgx* create_adfgx(const char* alphabet, const char* symbols, const char* dictionary, const char* transposition)`: Maakt geheugen aan voor een nieuwe ADFGX-encoder met dezelfde signatuur als in `adfgx.h`. De parameters worden diep gekopieerd met behulp van de functie `deep_copy`, omdat de pointers na de oproep in de main-functie niet langer beschikbaar zullen zijn. Controleer of alle allocaties geslaagd zijn en geef de resulterende pointer terug;

- `void set_transposition(adfgx* m, const char* transposition)`: Vervangt de huidige transpositiesleutel door een nieuwe, en voert een reallocatie uit indien nodig;
- `char* encode_simple(adfgx* m, const char* text)`: Encodeert een gefilterde boodschap op basis van de string dictionary. Er wordt voldoende geheugen aangemaakt (houd rekening met het dubbele aantal karakters), wat karakter per karakter ingevuld wordt. Het resultaat van `encode_simple(adfgx* m, "WAPENS")` met de settings zoals meegegeven in `adfgx.txt` is gelijk aan "AGAXGXXXFAXG";
- `char* decode_simple(adfgx* m, const char* encoded)`: Decodeert een gecodeerd bericht. Het resultaat van `decode_simple(adfgx* m, "AGAXGXXXFAXG")` is gelijk aan "WAPENS";
- `char* encode_hard(adfgx* m, const char* text)`: Encodeert een gefilterde boodschap op basis van de string dictionary en de transpositiesleutel `transposition`. Eerst wordt de inputstring gecodeerd met behulp van `encode_simple`, waarna spaties toegevoegd worden zodat de totale lengte van het gecodeerde bericht overeenkomt met een veelvoud van de lengte van de transpositiesleutel. Hierna worden de kolommen gerangschikt volgens de gesorteerde versie van de transpositiesleutel, en wordt het resultaat hiervan getransponeerd. De ingevoegde spaties kunnen nu eenvoudig weggefilterd worden om te leiden tot de gewenste encoding;
- `char* decode_hard(adfgx* m, const char* encoded)`: Decodeert een gecodeerd bericht. In het decoderingsproces moeten de spaties allereerst op de juiste plaats ingevoegd worden. Hierna wordt de bekomen matrix getransponeerd en worden de kolommen terug verwisseld. De spaties worden hierna weggefilterd en het resultaat wordt aangelegd aan de functie `decode_simple`, waarna het gedecodeerde bericht verkregen wordt;
- `char* brute_force(adfgx* m, const char* encoded, const char* start)`: Deze functie tracht de transpositiesleutel en de gedecodeerde versie van een bericht te achterhalen wanneer een ADFGX-machine aangetroffen wordt waarvan het ondersteunde alfabet en de overeenkomstige vertaling reeds gekend zijn. Hiertoe zal een groot aantal mogelijkheden getest worden, waarbij ervan uitgegaan mag worden dat de transpositiesleutel maximaal zeven (unieke) karakters bevat. Maak hierbij zeker gebruik van het aangeleverde bestand `permutations.txt`, dat alle unieke permutaties bevat. Een gecodeerde tekst werd verder voorzien in `encoded.txt`, en de bedoeling is om deze zonder kennis van de initiële posities toch succesvol te vertalen. Hiertoe gaan we brute-force tewerk: elk van deze permutaties wordt aangelegd als transpositiesleutel, die vervolgens gebruikt wordt om de versleutelde tekst te decoderen. Wanneer deze tekst een opgegeven kernwoord bevat (in dit geval "STRENGGEHEIM"), mogen we ervan uitgaan dat de juiste sleutel gevonden werd. Dit is een techniek die ook gebruikt werd bij het kraken van de Enigma-codering in de Tweede Wereldoorlog: Britse onderzoekers wisten namelijk dat Duitse berichten vaak constructies als "An die Gruppe" en "Streng geheim" bevatten, en er vaak ook mee begonnen. Zorg ervoor dat wanneer de gezochte sleutel en de overeenkomstige vertaling (door een implementatiefout in de decoder) niet werd gevonden, het gealloceerde geheugen toch vrijgegeven wordt en NULL teruggegeven wordt;
- `void free_adfgx(adfgx* m)`: Deze functie geeft alle geheugen, ingenomen door de ADFGX-encoder, weer vrij.

3 Aandachtspunten

Tracht bijzondere aandacht te hebben voor volgende zaken:

- Efficiënt geheugenbeheer: zorg ervoor dat er niet meer geheugen gebruikt wordt dan noodzakelijk. Het is echter niet de bedoeling om op bit- of byte-niveau geheugenbesparingen te voorzien (bijvoorbeeld door het combineren van twee kleine gehele waarden in één int). Controleer steeds of de geheugenallocatie en het openen van de nodige bestanden wel succesvol was;
- Het vermijden van memory leaks: raadpleeg Appendix B van deze opgave voor manieren om je code te testen;
- Onderhoudbaarheid van de code: tracht bestaande functies te hergebruiken, in plaats van code te kopiëren.

Appendix A: Uitlezen en wegschrijven van tekst

```
#include <stdio.h>
#include <stdlib.h>
#include <crtdbg.h>

int main (int argc, char** argv) {
    // Open het bestand met de in te lezen tekst
    FILE* file = fopen("read.txt", "r");
    if (!file) {
        printf("Fout bij het openen van het bestand!\n");
        exit(1);
    }
    // Alloceer geheugen voor 255 karakters + EOS
    char* line = malloc(sizeof *line * 256);
    // Controleer of de allocatie succesvol was
    if (!line) {
        printf("Fout bij het alloceren van geheugen!\n");
        exit(1);
    }
    // Lees de tekst in (let op: er zijn minstens 255 karakters)
    fread(line, 1, 255, file);
    // Beëindig de string
    line[255] = '\0';
    // Print ingelezen tekst
    printf("Ingelezen lijn: %s\n", line);
    // Sluit het bestand
    fclose(file);
    // Open het bestand waar de tekst naar weggeschreven wordt
    file = fopen("write.txt", "w");
    if (!file) {
        printf("Fout bij het openen van het bestand!\n");
        exit(1);
    }
    // Schrijf de ingelezen tekst weg
    fprintf(file, "%s", line);
    // Sluit het bestand
    fclose(file);
    // Geef het gealloceerde geheugen vrij
    free(line);
    // Return
    return EXIT_SUCCESS;
}
```


Appendix B: Geheugenbeheer

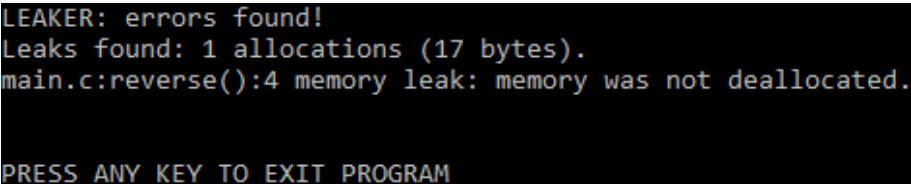
In het meegegeven headerbestand, `adfgx.h`, wordt `leaker.h` geïncludeerd. Deze library zal controleren of, na het uitvoeren van het programma, alle gealloceerde geheugen vrijgegeven werd. Indien dit niet het geval is, zoals bijvoorbeeld in volgend codefragment, wordt er een foutmelding weergegeven:

```
#include "leaker.h"

char* reverse(const char* input) {
    int l = strlen(input);
    char* output = malloc(sizeof *output * (l + 1));
    if (!output) {
        printf("Fout bij het alloceren van geheugen!\n");
        exit(1);
    }
    for (int i = 0; i < l; i++) {
        output[i] = input[l - 1 - i];
    }
    output[l] = '\0';
    return output;
}

int main (int argc, char** argv) {
    char* input = "Dit is een test!";
    printf("Input: %s\n", input);
    char* output = reverse(input);
    printf("Output: %s\n", output);
    return EXIT_SUCCESS;
}
```

Uit onderstaande foutmelding leren we dat het geheugen, gealloceerd in regel 4 van `main.c`, niet vrijgegeven wordt. Dit kan opgelost worden door `free(output)`; uit te voeren vooraleer het programma te beëindigen.



```
LEAKER: errors found!
Leaks found: 1 allocations (17 bytes).
main.c:reverse():4 memory leak: memory was not deallocated.

PRESS ANY KEY TO EXIT PROGRAM
```

Figuur 2: Een voorbeeld van een foutmelding bij een memory leak.