TP n° 24 : Stéganographie

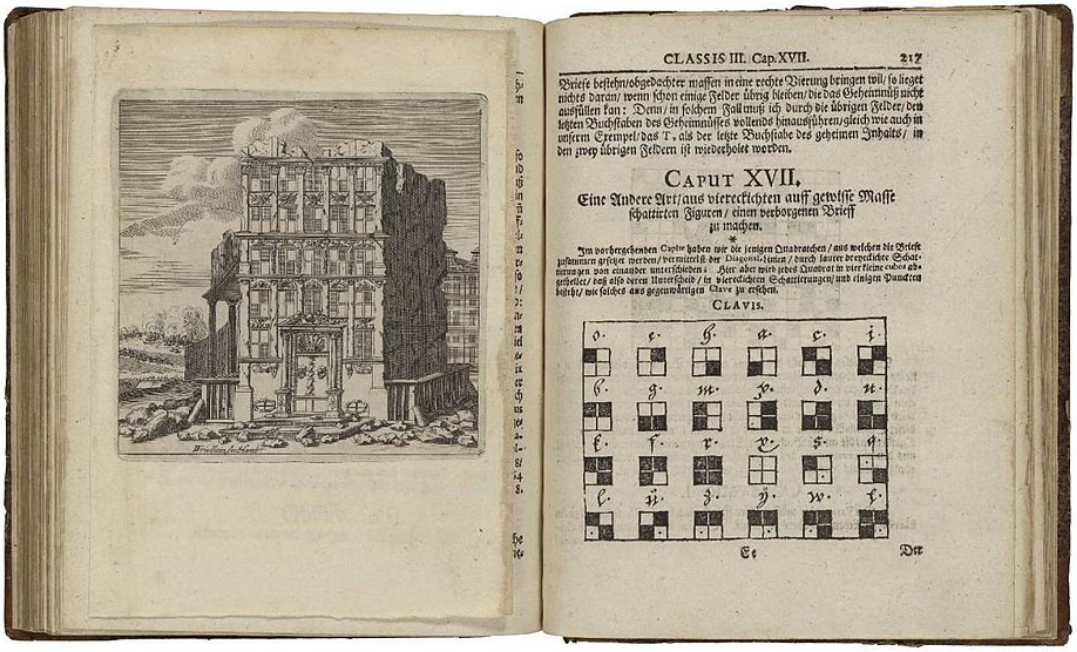


FIGURE 1 – Dans son ouvrage *Cryptographia oder Geheime Schrifft-münd-und Würkliche Correspondentz* (1684), JOHANNES BALTHASAR FRIDERICI nous montre un dessin apparemment anodin (page de gauche) mais qui contient en réalité un message secret. Ce dernier est codé par les fenêtres de l'immeuble : WIR HABEN KEIN PULVER MEHR (nous n'avons plus de poudre) avec la correspondance visible sur la page de droite.

La *stéganographie* est l'art de la dissimulation, celui de cacher un objet dans un autre. On peut par exemple vouloir cacher un message secret dans une image, ou même, une image dans une autre. Dans ce TP, on se propose de découvrir le message caché dans une image, puis de fabriquer soi-même des messages cachés. Enfin, il s'agit de répondre à la question posée par la figure 2.

1 Le format PGM

Le format d'image que nous allons utiliser est le format PGM ASCII pour *portable graymap file format* qui permet de représenter de manière très simple, mais non compressée, des images en niveaux de gris. Un fichier PGM comporte :

- Une première ligne avec obligatoirement les deux caractères « P2 », ce qui permet d'indiquer le format de l'image ;
- La *largeur* puis la *hauteur* de l'image, séparées par un caractère d'espacement ;
- La valeur maximale utilisée pour coder les niveaux de gris (dans ce TP ce sera toujours $2^{16} - 1 = 65535$) ;
- La valeur associée à chaque pixel de l'image, ligne par ligne, de gauche à droite, de haut en bas.

FIGURE 2 – Quelle est la meilleur équipe de go ? Terminer le TP et vous le saurez !

Chaque niveau de gris est ainsi codé par une valeur entre 0 et la valeur maximale, proportionnellement à son intensité. Un pixel noir est codé par la valeur 0, un pixel blanc est codé par la valeur maximale.

Par exemple, l'image au format PGM suivante

```
P2
24 7
15
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  3  3  3  3  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0 15  0
0  3  3  3  0  0  0  7  7  7  0  0  0 11 11 11  0  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0  0  0
0  3  0  0  0  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

correspond à l'image suivante

Les quatre premières lignes de l'image `image_avec_secret.pgm` sont ainsi :

```
> head -4 image_avec_secret.pgm
P2
1224 918
65535
51962 51963 52218 52402 52658 52 ...
```

On supposera dans toute la suite que les fichiers images ne contiennent pas de ligne de commentaire, qui sont des lignes qui peuvent commencer par un croisillon #.

2 Message textuel caché dans une image

2.1 Lecture en mémoire d'une image

Ce TP est entièrement à réaliser en utilisant le langage C. On propose le modèle de structure suivant pour représenter une image au format PGM ASCII. On rappelle que le type **uint16_t**, défini dans l'en-tête <stdint.h> désigne un entier non signé sur 16 bits, donc précisement une valeur entre 0 et 65536.

C

```c
struct image {
  int larg;
  int haut;
  int maxc;
  uint16_t **pixels;
};

typedef struct image image;
```

Question 1

Écrire une fonction de prototype `image *charger_image(char *nom_fichier)` qui prend en paramètre le nom d'un fichier et qui renvoie un pointeur vers une zone mémoire contenant les données de l'image que l'on aura allouée. On suppose que l'image est bien au bon format (on pourra utiliser des assertions pour le garantir). On gérera le cas où le programme n'arrive pas à lire le fichier et on renverra alors un pointeur `NULL`.

☞ **Remarque 1**

Comment lire un entier non signé 16 bits ? La solution la plus simple est de lire un entier avec le spécificateur de conversion `%d` *dans une variable temporaire de type* **int**, *de vérifier que sa valeur est bien entre 0 et 65535 puis de la transtyper vers le type* **uint16_t**. *Une autre solution est d'utiliser le spécificateur de conversion* `%lu` *pour* **unsigned short**, *mais cela suppose être certain que les entiers cours non signés sont exactement sur 16 bits. Pour les plus aguerris, une « bonne » solution, mais qui me semble complètement hors programme est d'utiliser la macro* `SCNu16` *definie dans l'en-tête* <inttypes.h>, *qui contient le bon spécificateur*

> *de conversion à placer après le* `%` *et d'utiliser le fait qu'en C la concaténation de deux chaînes de caractères se fait par simple juxtaposition. Pour les plus aguerris, si* `n` *est un* **`uint16_t`**, *on peut donc écrire directement* `fscanf(stream, "%" SCNu16, &n)`.

2.2 Cacher une lettre

L'image `image_avec_secret.pgm` contient un message caché de la manière suivante :

- chaque ligne de l'image permet de cacher un caractère ;
- dans une ligne donnée, chacun des 8 premiers pixels permet de cacher un bit qui est donné par le bit de poids faible du pixel ;
- ces bits sont rangés du bit de poids fort au bit de poids faible du caractère caché.

Ainsi une image qui contiendrait dans la première ligne les niveaux de gris suivants pour ses pixels :

```
80 81 81 89 89 90 89 80 ...
```

encoderait dans cette ligne la suite de bits de poids faibles :

```
01111010
```

qui encode le caractère `'z'` de code ASCII 122 qui s'écrit en effet 01111010 en binaire.

Question 2

Comment obtient-on le bit de poids faible d'un entier ?

Question 3

Donner un exemple de (début de) ligne de niveaux de gris qui permettrait d'encoder le caractère `'\0'`.

Question 4

Écrire une fonction de prototype **`char`** `caractere(`**`uint16_t`** `*tab)` qui prend en paramètre un (pointeur vers un) tableau d'entiers supposé de longueur au moins 8 et qui renvoie le caractère caché, avec la méthode décrite ci-dessus, dans ce tableau.

2.3 Lire le message secret

On suppose donc qu'une image qui cache un message est de largeur supérieure à 8, pour que chaque ligne puisse effectivement encoder un caractère. On suppose également que le message se termine quand on rencontre le caractère `'\0'` de code ASCII 0 encodé dans une ligne, et donc que ceci arrivera avant la fin de l'image.

Question 5

Écrire une fonction `int sauvegarder_message(image *img, char *nom_sortie)` qui prend en paramètres une image en mémoire et un nom de fichier et écrit le message caché dans l'image dans le fichier correspondant. On suppose que l'image, et le flux sont valides et en particulier non `NULL`. On suppose également que le message est valide et donc en particuler que `img->larg >= 8`. On gèrera le cas où le fichier ne peut pas être ouvert en écriture. La fonction renvoie un code de retour 0 si tout s'est bien passé et 1 sinon.

Question 6

Quel est le message secret caché dans l'image `image_avec_secret.pgm` ?

2.4 Cacher un caractère

Question 7

Écrire une fonction `void inserer_caractere(char c, uint16_t *tab)` qui prend en argument un caractère à cacher et un tableau d'entiers d'au moins 8 cases et qui cache le caractère dans le tableau en suivant les règles décrites ci-dessus.

FIGURE 3 – L'image dans laquelle un message secret a été caché.

2.5 Coder ses propres messages

Question 8

Écrire une fonction de prototype `int cacher(image *img, char *nom_entree)` qui prend en argument une image en mémoire et un nom de fichier que l'on suppose être un fichier texte, et qui cache le texte contenu dans ce fichier dans l'image. On supposera que l'image est de hauteur et de largeur suffisante. On n'oubliera pas la sentinelle `'\0'` finale. On gèrera toujours les cas d'erreurs et la fonction doit renvoyer un code de retour indiquant si tout s'est bien déroulé ou non.

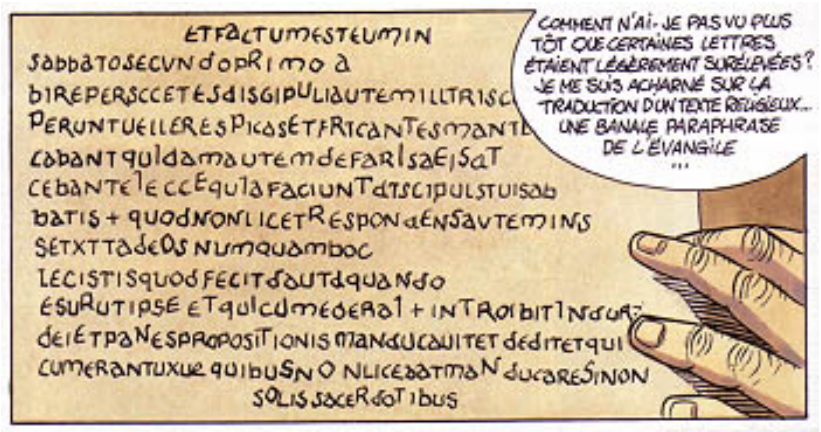
2.6 Sauvegarder une image

Question 9

Écrire une fonction `int sauvegarder_image(image *img, char *nom_fichier)` qui prend en paramètres une image en mémoire et un nom de fichier (qui n'existe pas forcément) et qui sauvegarde l'image au format PGM ASCII dans ce fichier.

Question 10

Cacher un message secret de votre cru dans l'image originale `lycee_champollion.pgm` et la sauvegarder sous le nom `message_de_<login>.pgm` où `<login>` est votre login utilisateur. Déposer votre image dans le répertoire partagé et demander à quelqu'un d'autre de trouver le message secret.



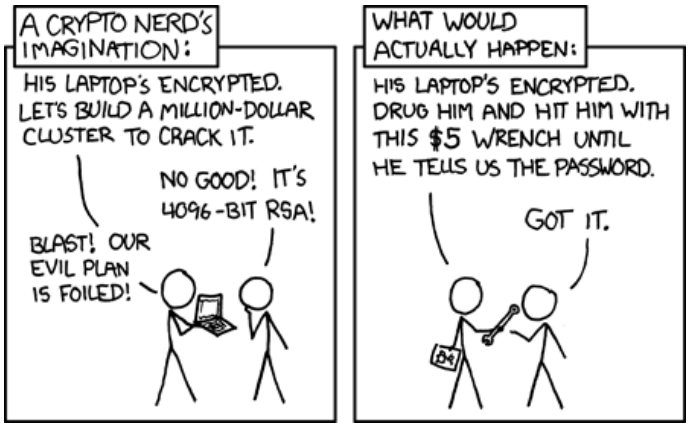
2.7 Fabriquer ses propres commandes

Question 11

Écrire un programme C complet qui prend en paramètres sur la ligne de commande trois ou quatre arguments :

- Le premier est soit `--trouver` soit `--cacher` selon que l'on veut *trouver* un message secret ou *cacher* un message secret ;
- Le deuxième est un nom de fichier correspondant à une image ;
- Le troisième est un nom de fichier correspondant à un fichier textuel ;
- Le quatrième, uniquement dans le cas `--cacher`, est un nom de fichier dans lequel enregistrer l'image modifiée contenant le message secret.

Si le premier argument est `--trouver`, ce programme doit trouver le message caché dans l'image donnée en deuxième argument et le sauvegarder dans le fichier textuel donné en troisième argument. Si le premier argument est `--cacher`, ce programme doit cacher dans l'image donnée en deuxième argument le message contenu dans le fichier textuel donné en troisième argument et sauvegarder le résultat dans le nom donné en dernier argument. On supposera que les images sont toujours de taille suffisante. Dans un vrai programme, il faudrait évidemment gérer très finement ce dernier point délicat, à l'origine de nombreuses failles de sécurité.



https://xkcd.com/538

3 Cacher une image dans une image

Nous avons vu comment cacher un message textuel dans une image. Il n'est pas très difficile de cacher une image entière B dans une image A. On suppose dans cette partie que les niveaux de gris sont codés sur 16 bits et donc que la valeur maximale est 65535. L'idée est simple : nous allons conserver les 8 bits de poids fort de l'image A pour représenter l'image A et utiliser les 8 bits de poids faible de l'image A pour encoder les 8 bits de poids fort de l'image B. L'image A et B seront donc légèrement dégradées puisque l'on n'utilise plus que les 8 bits de poids fort, mais en général, cela ne se voit pas.

Question 12

Répondre à la question posé par la figure 2 : *quelle est la meilleure équipe de go ?* de l'image `quelle-est-la-meilleure-equipe-de-go.pgm` ?

Question 13

Écrire un programme qui permet de cacher une image dans une autre en utilisant la méthode ci-dessus.

Question 14

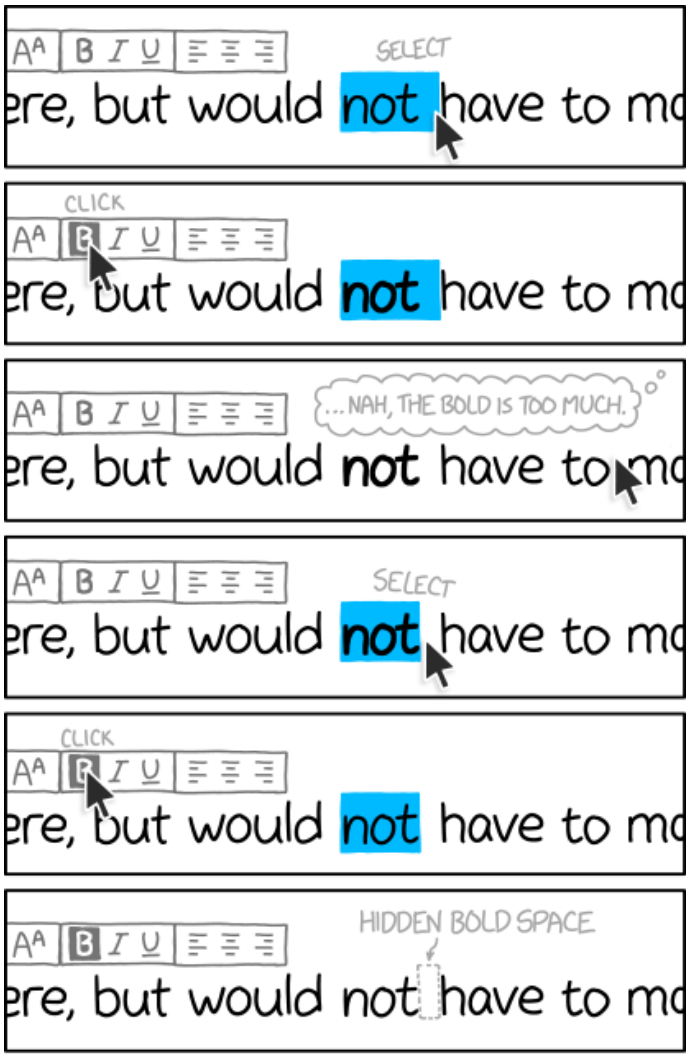
Cacher la meilleure équipe de go dans dans le lycée Champollion.

4 Bonus

Le format PPM permet de représenter des images en couleurs. Vous pouvez regarder la description du format sur la page WIKIPÉDIA : https://fr.wikipedia.org/wiki/Portable_pixmap#PPM.

Question 15

Cacher un message et/ou une image dans une image en couleur.



https://xkcd.com/2109/