Département IMA / 3A (S5)
http://laure.gonnord.org/pro/teaching/

Programmation Structurée 2012/2013
Sujet proposé par L. Gonnord
sur une idée de T. Maurice (IMA3)

Premier Projet de Développement Logiciel en C

Stéganographie : cacher du texte dans une image

Lire le sujet COMPLÈTEMENT dès la première séance !

Objectif

Ce projet de programmation structurée, a pour objectif de réaliser *en binôme*, un premier logiciel avec les notions acquises au S5 (et uniquement celles-ci).

Dans ce projet, vous mettrez en oeuvre les notions vues en cours de Programmation Structurée (conception, algorithmique, développement, critiques et documentation). Une partie du code vous sera fournie sous forme de bibliothèque écrite par un développeur tiers (Jérémie Dequidt)

Le code que vous nous demandons est réalisable avec des tableaux de taille statique (fixée à l'avance).

1 Plantons le décor !

Pour cela, regardons la page de Wikipédia http://en.wikipedia.org/wiki/Steganography

> Steganography is the art and science of writing hidden messages in such a way that no one, apart from the sender and intended recipient, suspects the existence of the message, a form of security through obscurity.

Nous allons donc jouer aux agents secrets, et coder du texte dans des images.

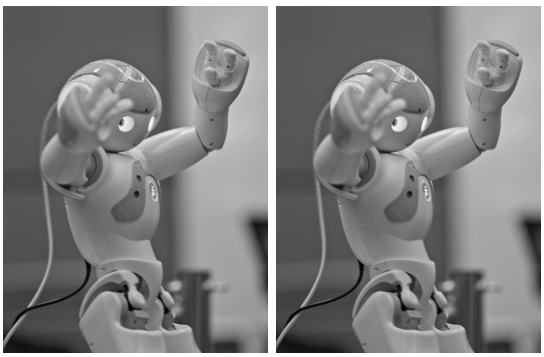

FIGURE 1 – L'image de référence et image modifiée en cachant le texte "tagada", vous voyez la différence vous ?

Pour cela, nous allons utiliser le fait que l'oeil ne repère pas de minuscules changements de teinte dans un pixel. Nous allons donc modifier certaines teintes de pixel en ajoutant des informations permettant de coder des caractères du texte à cacher.

1.1 Explications sur les images/photos

Stockage informatique des images La représentation la plus simple et la plus utilisée pour stocker des images en informatique est ce qu'on appelle une carte de pixels (en anglais *pixmap* / *pixelmap* mais bien souvent on utilise bitmap par abus de langage). Un pixel (abréviation de *picture element*) représente la plus petite unité que l'on peut afficher sur un écran. Le principe d'une carte de pixels est d'associer une couleur à chacun de ces pixels (dans un tableau). Il n'est pas nécessaire de stocker les coordonnées de chaque pixel puisqu'elles sont facilement accessibles lorsque l'on connaît les dimensions de l'image. Ainsi si notre image possède une largeur L et une hauteur H les couleurs de chacun des pixels seront stockées dans un tableau à une dimension (= un vecteur) où les L premières valeurs représenteront les couleurs de la première ligne de pixels, les L valeurs suivantes représenteront les couleurs de la deuxième ligne...

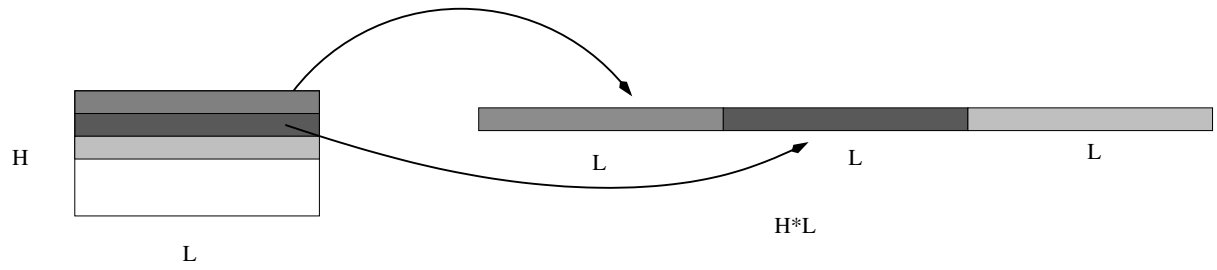

FIGURE 2 – Codage d'une image dans un vecteur

Couleur d'un pixel Le codage d'une couleur est stocké sur un ou plusieurs octets. Le format que nous utiliserons pour ce projet est une image noir et blanc où l'intensité de blanc est codée sur 1 octet (0 = pixel "noir", 255 = "pixel blanc" codé par un `unsigned char` en C). Dans la suite du sujet et par abus de langage, nous utiliserons "couleur" pour désigner l'intensité de blanc d'un pixel.

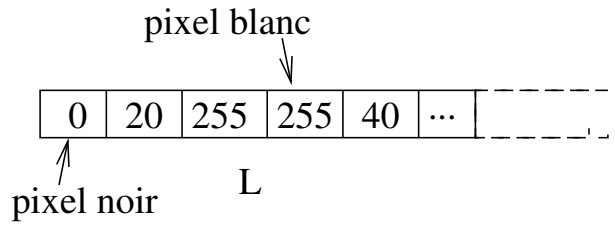

FIGURE 3 – Codage d'une image N/B : chaque case du tableau correspond à un pixel

1.2 Travail à réaliser

Réaliser un logiciel qui, à partir d'une image et d'un texte, dissimule le texte dans l'image. Le logiciel saura aussi retrouver le texte à partir de l'image.

Le logiciel n'affichera ni l'image d'entrée ni l'image de sortie. Pour visualiser une image, on utilisera par exemple l'utilitaire `gqview` ou `eog`.

Le principe général pour réaliser cette opération est de lire l'image pixel par pixel, et de générer une nouvelle image contenant les pixels modifiés. Chaque caractère du texte à cacher (représenté par 8 bits) sera codé dans 4 pixels successifs de l'image (2 bits du caractère codé dans chacun des 4 pixels).

EXEMPLE 1 *Prenons le caractère t, son code ascii est* 116*, qui s'écrit en binaire* 01110100*. On va donc cacher* 01 *dans un premier pixel, puis* 11*, puis* 01 *puis* 00*. Les* 4 *premiers pixels de l'image fournie* `nao.png` *sont* $90, 87, 90, 94$*. Prenons* $90 = (01011\mathbf{10})_2$ *base 2, il sera modifié en remplaçant les bits en gras* 10 *par* 01 *(les deux bits de poids forts du code ascii de t). Le premier pixel deviendra donc* $(01011\mathbf{01})_2$*, c'est-à-dire* 89*. Les trois pixels suivants (faire les calculs) seront* 87*,* 89 *et* 92*.*

Le codage du texte commencera dès le premier pixel de l'image. Il faudra ajouter à la fin du codage un caractère spécial qui permettra au programme décodeur de détecter la fin.

Nous ne détaillons pas d'avantage, à dessein. C'est à vous de réaliser cet algorithme de bout en bout, d'effectuer le découpage en sous tâches, ... Vous pouvez bien évidemment vous servir du Web comme source, notamment en ce qui concerne le découpage en bits.

Cahier des charges

1. (Base) Réaliser la fonctionnalité décrite plus haut. À titre indicatif, notre solution fait moins de 150 lignes (commentaires inclus).
2. Faire en sorte que l'utilisateur puisse choisir son image, le texte à inclure, ...et que l'interface *textuelle* soit ergonomique.
3. Prendre en considération l'aspect performance, éventuellement en réalisant plusieurs versions.
4. Modifier votre algorithme pour cacher votre texte ailleurs qu'au début de l'image. Il faudra que le destinataire de l'image récupère donc l'information de début de codage.
5. (Bonus) Essayer de cacher une (petite) image dans une grande image, essayer de détecter si un fichier a été modifié (analyse statistique), ...

1.3 Figures Imposées

Pour réaliser votre tâche, vous allez vous appuyer sur :
- Une bibliothèque écrite par un développeur tiers (J. Dequidt). Le code correspond à un ensemble de fonctions qui vous seront utiles pour manipuler des images png : chargement d'une image, création d'un tableau de taille quelconque pour stocker des pixels, et sauvegarde d'une image. On vous fournit un Makefile qui permet de lier votre code à la bibliothèque.
- Vos connaissances acquises en Programmation Structurée. Aucune autre connaissance n'est requise.

2 Modalités de travail

Vous avez deux séances de TP (séances 9 et 10) pour avancer au maximum le projet, puis ensuite 3 semaines de travail *en binôme*.

> Pour travailler chez vous, il faudra faire en sorte de récupérer vos fichiers de l'extérieur. C'est expliqué sur le Twiki. Ou alors, prévoyez une clef USB. Faites aussi en sorte que les deux membres du binôme aient le code courant...

2.1 Première séance : TP numéro 9

Préliminaires Avant de commencer, chaque binôme va récupérer le source du sujet (quelques fichiers sources et autres choses utiles) sur la page web du cours, désarchive ce répertoire.

Makefile et utilisation des librairies Dans la première séance, on vous demande de tester la librairie fournie. Un Makefile est contenu dans l'archive et un fichier nommé `stega.c` est prêt à être complété. Tester les principales fonctions de la librairie, notamment en ouvrant un fichier PNG et en le sauvant sous un autre nom. Se reporter au fichier d'entête (`include/algo_ima14.h`) pour l'explication des fonctionnalités de la librairie. Il est important de bien comprendre les fonctions de librairie avant de s'en servir.

Compréhension du sujet Faire un premier bilan des étapes que vous devez réaliser et faites en part à un encadrant afin que nous puissions vérifier la bonne compréhension du sujet. Faites des dessins pour montrer l'idée de votre algorithme.

2.2 La suite !

La suite est en autonomie, on ne vous guide plus ! Vous devez faire les analyses, les algorithmes, bien concevoir vos algos avant de les coder. Cette partie est aussi évaluée. Prenez l'habitude de bien commenter au fur et à mesure, et de commenter en ANGLAIS.

Attention à bien tester *chacune* de vos fonctions, en vérifiant bien leur résultat sur des exemples bien choisis. On ne bâtit pas sur de mauvaises fondations. Il est inutile d'avancer si une fonction ne fonctionne pas.

Annexe - Consignes pour le rendu

À lire, relire, et rerelire !

A Modalités du rendu de projet

Nous récupérerons **le jeudi 10 janvier 2013 à 18h** (5 points par jour de retard, 2 points dès la première heure) vos projets sous Moodle. Une unique archive nommée `Nom1_Nom2.tgz` (`Nom1` et `Nom2` les noms des deux membres du binôme dans l'ordre alphabétique) sera déposée sur Moodle.

L'archive tgz devra se décompresser en un répertoire nommé `Nom1_Nom2` contenant :

- un fichier `Readme.txt` décrivant rapidement votre logiciel, ses fonctionnalités et donnant un mode d'emploi succinct.
- un répertoire `Code` (avec Makefile). Pour faciliter la correction, le binaire s'appellera `mosaique`.
- un répertoire `Images` dans lequel les fichiers images seront déposés ou générés par votre programme.
- un fichier `Nom1_Nom2.pdf` contiendra votre rapport. Le rapport ne comprendra pas plus de 5 pages, devra être clair et précis et notamment comporter les limitations de votre outil. **PAS de rapport papier SVP !**

Attention ! votre archive devra être propre, ie ne pas comporter de fichier .o, tilde, binaire, etc.

Nous fournirons un script qui permettra de vérifier les consignes. Des points seront enlevés aux binômes pour lesquels le script renvoie une erreur.

B Le compte-rendu

Généralités :

- Réécrire le sujet ne sert à RIEN.
- Ce n'est pas la peine de mettre les codes en annexe
- Ce n'est pas la peine d'imprimer votre rapport
- Le compte-rendu est en pdf et pas en autre format
- Rapport en 11 pt, interligne simple, sans fioriture (pas de titre en couleur), maximum 5 pages A4.
- La grammaire et l'orthographe seront corrects.
- La grammaire et l'orthographe seront corrects.
- Vous identifierez **clairement** les points du cahier des charges qui ont été clairement traités et ceux qui manquent.
- Les difficultés que vous avez rencontrées seront décrites.
- Les algorithmes pourront par exemple être décrits selon le modèle suivant (en ajoutant des dessins si nécessaire). Le pseudo code est en général inutile pour des algorithmes de base.

Exemple de description de fonction

`int chargeImageReference(char * fileName, unsigned char ** bufferRef, int * w, int * h)`

- **Spécification** :
  - charge une image de type PNG (fileName) dans un vecteur de char (bufferRef) qui est supposé déjà alloué.
  - stocke la taille de l'image dans w et h.
  - retourne 1 si tout c'est bien passé, 0 sinon.
- **Conception/étapes** :
  - `fileName` est un `char *` : chaîne de caractères de taille non fixée à l'avance, ce nom n'est pas modifié.
  - `bufferRef` est un pointeur vers le Tableau résultat : chaque case du tableau contiendra à la fin une valeur entre 0 et 255 (niveau de gris du pixel)
  - étapes : ouverture du fichier avec erreur 0 si il n'existe pas, vérifications : fichier de bonne taille (et récupération de la hauteur et largeur dans w et h), de bon type, et passage en niveaux de gris (si le fichier est en couleur), ensuite récupération des pixels et fermeture du fichier.

C Le code lui-même !

- Les noms des fonctions, des identifiants, les commentaires seront faits en langue anglaise, histoire de s'habituer.
- Les codes seront indentés avec indent -kr .
- Les fichiers comporteront les noms des binômes (en commentaires).

La pompe/triche sera lourdement sanctionnée. C'est un travail en binôme, pas en classe entière.

D Modalités d'évaluation

Nous évaluerons en plus du *respect des consignes*, la maîtrise des concepts de Programmation Structurée, ainsi que la qualité de votre développement et de votre programme :

- la conception, c'est à dire l'analyse papier, le choix du découpage en fonctionnalités, et les explications de vos algorithmes.
- les commentaires, la lisibilité du code, l'indentation ;
- la *simplicité* ;
- l'efficacité du code et la pertinence des évaluations de performance ;
- le rendu final et l'esthétique des exemples ;
- **la gestion de ces différents points durant les séances de TP sera aussi évaluée**

Évidemment, cette liste n'est pas exhaustive !