INFO0030 : Projet de Programmation
Gestion de Code-Barres

B. Donnet, E. Marechal
Université de Liège

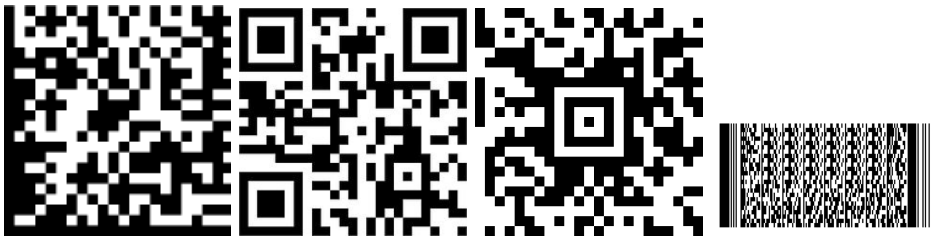
Alerte : Évitez de perdre bêtement des points...

- Nous vous conseillons **vivement** de relire et d'appliquer les **guides de style et de langage**, mis à votre disposition sur eCampus (INFO0030, Sec. Supports pour le Cours Théorique). Cela vous permettra d'éviter de nombreuses erreurs et de perdre des points inutilement.
- Nous vous conseillons de consulter la **grille de cotation** utilisée pour la correction des projets afin d'éviter de perdre bêtement des points. Elle est disponible sur eCampus (INFO0030, Sec. Procédures d'Évaluation).
- Votre code sera compilé et testé sur les **machines CANDI qui servent de référence**. La procédure à suivre pour se connecter en ssh aux machines CANDI est disponible sur eCampus. Veillez donc à ce que votre code fonctionne dans cet environnement.

1 Contexte

On rencontre de plus en plus souvent des codes-barres carrés à deux dimensions. Vous en avez peut-être déjà vu sur des colis postaux, sur des panneaux publicitaires, des cartes de visite, des certificats, des pièces électroniques, des tickets de concert, ... Ils sont même utilisés dans le domaine artistique. Ainsi, le clip vidéo de la chanson *Integral* des Pet Shop Boys [1] fait, par exemple, passer plusieurs liens via des QR Code. [2] La police de New York utilise aussi ce système pour charger directement les informations des conducteurs à partir de leur permis de conduire dans leur système TraCS. [3]

Ces code-barres sont capables de stocker une certaine quantité d'information utile à laquelle est ajoutée de l'information redondante qui permet au code-barres d'être plus robuste. En effet, même si une partie est altérée (on peut imaginer que la pluie a endommagé un ticket de concert), le lecteur de code-barres sera quand même capable de lire le message original en exploitant l'information redondante.



(a) Datamatrix (b) QR Code (c) Aztec (d) PDF-417

FIGURE 1 – Exemples de format de code-barres.

1. Pet Shop Boys, "Fundamental", 2006, chez Parlophone Rhino.
2. *Quick Response Code*. Voir http://en.wikipedia.org/wiki/QR_code.
3. *Traffic & Criminal Software*. Voir http://criminaljustice.state.ny.us/crimnet/ojsa/initiatives/tracs.htm

Il existe de nombreux formats utilisés en pratique. La Fig. 1 donne différents exemples de format qu'on peut décoder avec le lecteur adéquat ou avec un smartphone, si celui-ci est équipé du logiciel idoine.

Ces codes-barres sont très pratiques et sont de plus en plus utilisés car ils permettent de stocker beaucoup d'informations sur une petite surface tout en étant robustes aux altérations.

L'Université de Liège, découvrant le potentiel de ces tags, décide de les utiliser pour améliorer sa gestion des cartes d'étudiant. En effet, lors des examens écrits, les cartes d'étudiant sont utilisées pour vérifier à la fois que l'étudiant est bien inscrit à l'examen (en vérifiant que son matricule ULiège se trouve bien dans une liste d'inscrits), qu'il est bien en ordre administratif et que l'étudiant est bien le propriétaire de la carte (via la photo). Les deux premières de ces vérifications peuvent très vite s'avérer fastidieuse, surtout lorsqu'on a affaire à plusieurs centaines d'étudiants. Dès lors, la décision a été prise d'encoder le matricule ULiège d'un étudiant sous la forme d'un code-barres 2D et de placer ce code-barres dans un coin de la carte d'étudiant. La carte peut alors être scannée rapidement via un petit lecteur portable qui affiche la photo de l'étudiant ainsi qu'une indication précisant s'il est inscrit à l'examen et s'il a le droit de le présenter (i.e., l'étudiant est en ordre administratif). Une liste des étudiants ne s'étant pas présenté à l'examen peut également être générée directement au terme de l'examen.

Dans ce projet, il vous est demandé de concevoir un prototype de décodeur de code-barres 2D utilisé pour encoder un matricule ULiège.

2 Format du Code-Barres

2.1 Généralités

Les codes-barres utilisés pour encoder les matricules ULiège sont de forme carrée et de taille 7×7. La Fig. 2 vous montre la forme d'un tel code-barres. La zone blanche est la zone qui contient le nombre entier (*zone nombre*, cfr. Sec. 2.2) et la zone grise est la zone qui contient les informations redondantes qui permettront de corriger une erreur dans les codes-barres (*zone de parité*, cfr. Sec. 2.3).

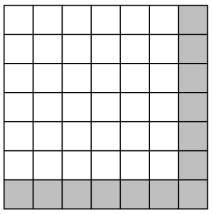


FIGURE 2 – Format du code-barres 2D.

2.2 Zone Nombre

Pour encoder un matricule ULiège dans un code-barres 2D, nous allons simplement encoder sa représentation binaire inversée [4] dans la zone nombre. Pour rappel, la notation binaire classique d'un entier est sa notation en base 2, c'est-à-dire constituée de 0 et de 1 qui correspondent respectivement à "blanc" et "noir" dans nos codes-barres.

Tout nombre naturel peut s'écrire en binaire, c'est-à-dire se décomposer en somme de puissance de 2 (2, 4, 8, 16, 32, ...). Par exemple, 83 peut se décomposer en

$$1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0.$$

et donc se noter `1010011` en binaire. [5]

4. Inversée dans l'ordre des bits, le nombre commençant par ses bits de poids faible au lieu de ceux de poids fort.
5. Plus de détails sur la conversion décimal/binaire se trouvent sur les slides 48 à 56 de l'Introduction du cours INFO0946.

Si l'on veut généraliser la notation binaire d'un nombre N représentable dans les 36 cases de la zone nombre de notre code-barres, on peut dire que chaque case correspondra à un a_i tel que $N = \sum_{i=0}^{35} a_i \times 2^i$.

Si l'on considère que les cases sont numérotées de gauche à droite et de haut en bas, nous noircirons la première case si et seulement si a_0 vaut 1, la seconde si et seulement si a_1 vaut 1 et ainsi de suite. De cette manière, nous représentons le chiffre 83 dans la zone "nombre" de notre code-barres comme indiqué à la Fig. 3.

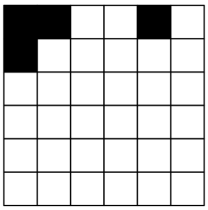


FIGURE 3 – La zone nombre pour "83" dans notre code-barres 2D.

Vu qu'un matricule ULiège est un nombre de 8 chiffres, la matrice sera bien plus remplie. Par exemple, le matricule 87651234 est représenté à la Fig. 4.

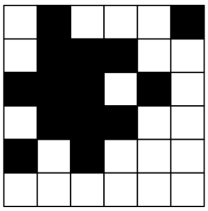


FIGURE 4 – La zone nombre pour "87651234" dans notre code-barres 2D.

2.3 Zone de Parité

Une fois les données placées, nous devons ajouter les informations redondantes qui permettront de vérifier si le code-barres a été altéré. Un *code de parité* va être utilisé pour cela.

Un code de parité se calcule en comptant le nombre de bits à 1 (noir dans notre cas). Si celui-ci est impair, le bit de parité vaut 1. Si par contre il est pair, il vaut 0. Notez que si l'on considère chaque case comme un entier valant 0 ou 1, la somme de parité correspond au reste de la division par 2 de la somme des différentes valeurs.

Dans notre code-barres 2D, un premier code de parité est calculé sur chacune des ligne et placé dans la dernière colonne. Un deuxième code de parité est calculé sur les colonnes et placé dans la dernière ligne. Enfin, la parité de la dernière ligne et de la dernière colonne (qui est nécessairement la même !) est stockée dans la case en bas à droite. La Fig. 5 reprend le précédent exemple de matricule ULiège complété de ses bits de parité (ici en gris mais ils seront, bien entendu, en noir dans les codes-barres réels).

3 Correction d'une Erreur

L'intérêt de la zone de parité est qu'elle permet de détecter et corriger une erreur sur un bit quelconque du code-barres pour autant qu'il n'y en ait qu'une seule. Les erreurs dans le code-barres peuvent être dues à diverses circonstances : une erreur à la lecture, une altération du code-barres due à la pluie, un trait à l'encre indélibile, ...

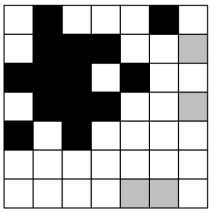


FIGURE 5 – Le code-barre 2D complet pour le matricule ULiège 87651234.

Pour comprendre comment corriger une erreur, nous allons séparer le problème en différents scénarios.

Pour chacun des scénarios décrits ci-dessous, il faut se demander quelle est la coordonnée du point défectueux si l'on considère qu'il n'y a qu'une seule erreur (la numérotation des lignes et des colonnes se fait à partir de l'indice 0).

— la parité de la ligne 2 et de la colonne 0 est mauvaise.
— la parité de la ligne 1 et du pixel inférieur droit est mauvaise.
— la parité du pixel inférieur droit est mauvaise.

A partir de ces trois scénarios, il doit être possible de déterminer les trois grandes "familles" d'erreurs uniques possibles.

Attention, si on sait qu'il y a une seule erreur dans le code-barres, on sait qu'on pourra toujours la corriger. Maintenant, ce n'est pas parce qu'on se retrouve dans une situation où il y a un seul bit de parité ligne et un seul bit de parité de colonne qui ne sont pas valides qu'il n'y a qu'une seule erreur dans la zone nombre.

4 Génération d'Images

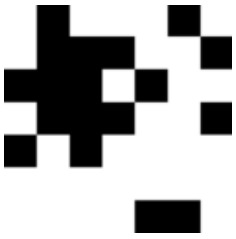


FIGURE 6 – Image au format `PBM` pour le matricule ULiège 87651234.

Lors d'un projet précédent, nous avons demandé d'implémenter une librairie permettant de manipuler des images au format `PNM` et, en particulier, le format `PBM`. Ce format convient parfaitement aux codes-barres représentant un matricule ULiège. Ainsi, il suffirait, simplement, de stocker le code-barres dans un fichier (avec un en-tête adéquat pour respecter le format `PBM` et chaque bit du code-barres représentant un caractère ASCII de l'image) pour avoir une image `PBM` toute faite.

Cependant, stocker tel quel le code-barres risque de rendre l'image illisible. Le code-barres tel qu'envisagé à la Sec. 2 formerait une image de taille 7×7, ce qui est beaucoup trop petit.

Dès lors, au moment de stocker le code-barres dans un fichier `PBM`, il devient nécessaire d'augmenter la résolution de l'image. Pour ce faire, au lieu de stocker directement la matrice 7x7, on va stocker une matrice 70×70. Ceci signifie que chaque bit (0 ou 1) du code-barres sera stocké par un carré de 10×10 bits. Ainsi, par exemple, les deux premiers bits du code-barres correspondant au matricule ULiège 87651234 (cfr. Fig. 4) seront représentés par [6] :

6. Le fichier `PBM` correspondant est disponible, à titre d'exemple, sur la page Web du cours

```
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
```

Le résultat complet de l'image est donné à la Fig. 6.

Dans le cadre de ce projet, nous vous demandons donc de réutiliser la librairie `PNM` pour la manipulation des images correspondant aux code-barres (cfr. Projet 1).

5 Enoncé du Projet

Il vous est demandé d'écrire un programme permettant d'implémenter la représentation d'un matricule ULiège sous la forme d'un code-barres tel que décrit à la Sec. 2.

Votre projet devra :

— être soumis dans une archive `tar.gz`, appelée `codebarre.tar.gz`, via la Plateforme de Soumission. La décompression de votre archive devra fournir tous les fichiers nécessaires dans le répertoire courant où se situe l'archive. Vous penserez à joindre tous les codes sources nécessaires.

— réutiliser la librairie PNM que vous avez réalisé dans un précédent projet. Il est évident que l'équipe pédagogique s'attend à ce que vous ayez apporté des modifications à votre librairie PNM en fonction du feedback donné. A cette fin, vous nous fournirez le code source nécessaire de votre librairie PNM.

— être modulaire, i.e., nous nous attendons à trouver un (ou plusieurs) header(s) et un (ou plusieurs) module(s). Il est évident que votre projet doit contenir un programme utilisant le code écrit.

— appliquer les principes de la programmation défensive (vérification des préconditions, vérification des mallocs, ...). Pensez à libérer la mémoire allouée en cours de programmation afin d'éviter les fuites de mémoire.

— être parfaitement documenté. Vous veillerez à documenter correctement chaque header/fonction/procédure/structure de données que vous définirez. Votre document suivra les principes de l'outil `doxygen`.[7]

— implémenter les fonctions permettant de corriger une éventuelle erreur dans les codes-barres (cfr. Sec. 3).

— être validé par une librairie de tests unitaires. Pour cela, vous utiliserez l'outil `seatest` vu au cours.[8] N'oubliez pas de joindre, dans votre archive, les fichiers `seatest.c` et `seatest.h`. Ces tests unitaires porteront sur votre librairie `PNM` mais aussi sur certaines fonctionnalités que vous aurez mis en place pour ce projet.

— comprendre un `Makefile` permettant au moins de

1. compiler votre projet. L'exécution de la commande

```
$>make
```

doit produire un fichier binaire, exécutable, appelé `codebarre`. Attention, lors de la compilation, vous ne pouvez pas recompiler les fichiers sources propres à la librairie `PNM`. A la place, vous devez intégrer la librairie `libpnm` (cfr. projet précédent). Par contre, vous devez fournir les fichiers sources pour cette librairie dans votre archive `tar.gz` afin que l'équipe pédagogique puisse vérifier que vous avez pris en compte le feedback donné.

2. Le fichier binaire généré doit pouvoir être exécuté de la façon suivante :

```
$>./codebarre -f <fichier> -o <dossier_output>
```

7. Vous veillerez, aussi, à documenter en `doxygen` votre librairie `PNM`.
8. Le code source de `seatest` est disponible sur la page Web du cours.

où `<fichier>` est le fichier texte contenant les matricules, et `<dossier_output>` est le dossier où les images doivent être écrites (si ce dossier n'existe pas, il doit être créé par votre programme). Le fichier contenant les matricules se présente de la façon suivante :

```
20100749
20111182
20103045
20105542
...
20110255
```

Le nombre de matricules dans ce fichier est indéterminé (le fichier pourrait très bien être vide). Il n'est pas certain non plus que le fichier est bien formé (il pourrait contenir autre chose que des matricules ou bien certains matricules sont malformés). Attention, votre programme doit *impérativement* accepter les chemins absolus et relatif des fichiers (ex : `/home/user/x/matricules.txt` ou bien `../../matricules.txt`)

Pour chaque matricule, une image au format `PBM` devra être générée (cfr. Sec. 4). Le nom de fichier, pour chaque image correspondra au numéro de matricule lu dans le fichier, avec l'extension `.pbm`.

3. générer de la documentation. L'exécution de la commande

```
$>make doc
```

doit produire une documentation, au format HTML, dans le sous-répertoire `doc/`.

4. générer un fichier exécutable de tests. L'exécution de la commande

```
$>make test
```

doit produire un fichier binaire, exécutable, de nom `codebarre_test` permettant de tester votre programme (tests unitaires). Ce fichier binaire devra pouvoir être exécuté de la façon suivante :

```
$>./codebarre_test
```

5. générer la librairie `libpnm` via l'exécution de la commande

```
$>make lib
```

Le résultat de cette commande est une libraire appelée `libpnm.a`

6. Tout Makefile qui se respecte doit aussi inclure la commande

```
$>make clean
```

qui permet de supprimer tous les exécutables et librairies (et autres fichiers) générés antérieurement par le Makefile.

Toute question relative au projet peut être posée sur le forum de la page web du cours. [9]

Tout projet ne compilant pas se verra attribuer la note de 0/20.

Pour vous aider à rédiger le meilleur code possible, référez-vous au **guide de langage** et au **guide de style** disponibles sur eCampus. La grille de cotation utilisée pour la correction des projets est également en ligne.

Il est impératif de respecter **scrupuleusement** les consignes sous peine de se voir attribuer une note de 0/20 pour non respect de l'énoncé.

9. cfr. eCampus