SAE IMAGE

Petite explication avant de passer au codage:

Dans cette SAE nous allons avoir besoin d'un editeur hexadecimale: okteta et d'un utilitaire (de la suite logicielle ImageMagick) très commode que nous utiliserons par la suite dans ce devoir: display

Voici comment lire une adresse dans un fichier okteta:

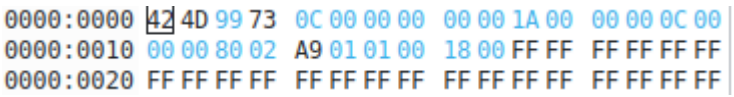
Les chiffres marqué à gauche sont les adresses. Donc 0000: 0000 est l'adresse 0x00 à cette adresse nous retrouverons l'octet qui a pour valeur 42. L'octet suivant est: 4D sont adresse est: 0x01

De la même maniere les valeurs 0000: 0010 correspond à l'adresse 0x10 à cette adresse nous trouverons l'octet 00. Les deux octets suivant sont 00 80 cela correspond à l'adresse 0x11 et l'adresse 0x12.

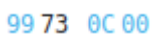
PARTIE A:

EXERCICE A0 :

Partie 1:



Ces deux premiers octets que nous pouvons lire dans ce fichier sont 42 4D codé en hexadécimal, cela correspond au type de fichier que nous sommes en train d'analyser (42 et 4D représente dans la table ASCII les lettres BM). Dans notre cas cela signifie donc que c'est un fichier BMP (image).



Ces quatre octets suivants correspondent à la taille du fichier. Ici la taille est représenter en little endian, il faut donc inverser les octets (donc 99 73 0C 00 va devenir 00 0C 73 99) lorsqu'on le calcule, cela nous donne 816 025 octets ce qui est la taille de notre fichier.

Les quatre octets suivants sont des champs réserver. (ils valent 00 00 00 00).

Puis les quatre octets suivants correspondent à l'adresse de la zone de définition de l'image, donc à l'adresse 0x1A commencent le codage des pixels. Il vaut: 00 00 00 1A en tenant compte de l'endiannes.

Tout ceci est l'en-tête du fichier Bmp que nous sommes en-train d'analyser. Il est codé sur 14 octets.

Maintenant nous allons analyser l'en-tête du bitmap. Sur 4 octets nous trouvons la taille en octets de cet en-tête. Dans ce fichier cela correspond à (0C 00 00 00). Sachant que nous sommes toujours en little endian nous devons inverser l'ordre des octets, cela devient (00 00 00 0C) qui vaut 12 octets. Donc la taille de l'en-tête du bitmap vaut 12 octets.

```
80 02 A9 01
```

Les 4 octets suivants sont la largeur de l'image en pixels, (80 02 A9 01). 01 A9 02 80 (en tenant compte de l'endiannes)= 27 853 440 pixels. (Pour le calculer nous faisons: 0 + 8 * 16 + 2 * 16^2 + 0 + 9 * 16^4 + 10 * 16^5 + 1 * 16^6+ 0). Donc notre fichier a 27 853 440 pixels de largeur.

```
01 00 18 00
```

Les 4 octets suivants (01 00 18 00) sont la hauteur de l'image en pixels. 00 18 00 01 = 1 572 865. Donc notre image a une hauteur de 1 572 865 pixels. (Pour le calculer nous faisons: 1 + 8 * 16^4 + 1 * 16^5)

Les octets suivants sont les octets de l'image rapellons nous qu'à l'adresse 0x1A nous obtenons l'adresse de la zone de définition de l'image et nous y sommes arrivé nous sommes à l'adresse 0x1A.

Partie2:

```
99 73 0C 00
```

Lorsque nous affichons l'image avec la commande display, nous obtenons une erreur. Cela est du à la taille du fichier. En effet dans le fichier la taille entrer est de 00 0C 73 99 qui vaut 816 025 octets.

Alors que lorsque nous faisons ls -l dans le terminal pour voir la taille du fichier cela nous indique 816 026.

```
816026 nov.  30 21:06 ImageExemple.bmp
```

Ce problème est du au fait que nous oublions de calculer l'octet à l'adresse 0. Expemple: de 0 à 5 il y a 5 valeurs: 1, 2, 3, 4 et 5 cela est compter par ce que l'on entre dans le fichier tandis que dans le terminal et pour la taille du fichier il faut compter de 0 à 5 nous trouvons donc les valeurs: 0, 1, 2, 3, 4 et 5.

Pour résoudre se problème nous devons donc ajouter 1 à 816 025 et le traduire en hexadécimal. Cela nous donne donc: 00 0C 73 9A. Sachant que nous devons entrer cette valeur en little endian nous allons inverser les octets, cela va nous donner 9A 73 0C 00. Maintenant nous pouvons entrer cette nouvelle valeur à la place de l'ancienne et le problème sera résolu.

Dans l'en-tête du fichier nous allons donc obtenir:

```
9A 73 0C 00
```

Voici donc la nouvelle en-tête que nous avons:

```
42 4D 9A 73 0C 00 00 00  00 00 1A 00 00 00 0C 00
00 00 80 02 A9 01 01 00  18 00 FF FF FF FF FF FF
```

EXERCICE A1:

Image0.bmp

```
0000:0000 42 4D 4A 00  00 00 00 00  00 00 1A 00  00 00 0C 00  BMJ.............
0000:0010 00 00 04 00  04 00 01 00  18 00 00 00  FF FF FF FF  ............ÿÿÿÿ
0000:0020 00 00 FF FF  FF FF FF FF  FF 00 00 FF  FF FF FF 00  ..ÿÿÿÿÿÿÿ..ÿÿÿÿ.
0000:0030 00 FF 00 00  FF FF FF FF  00 00 FF FF  FF FF FF FF  .ÿ..ÿÿÿÿ..ÿÿÿÿÿÿ
0000:0040 FF 00 00 FF  FF FF FF 00  00 FF                    ÿ..ÿÿÿÿ..ÿ
```

L'en-tête du fichier nous est donné dans la consigne. Nous allons donc détailler directement le contenu.

```
00 00 FF
```

Le premier pixel de notre fichier commence à l'adresse 0x1A et il représente le pixel en bas à gauche. Ce pixel est codé sur 3 octets, un octet pour la couleur bleu allant de 0 à 255, un octet pour vert allant de 0 à 255 et un octet pour le rouge allant de 0 à 255 (en tenant compte de l'endian car si on ne tient pas compte de l'endianesse on obtient Rouge, Vert, Bleu). Sachant que ce pixel doit être de couleur rouge nous auront 00 00 FF (qui représentent le rouge et qui est codé en hexadecimal) (FF=255)

```
FF FF FF
```

Le second pixel est de couleur blanche donc nous aurons FF FF FF. Nous allons continuer à coder comme cela j'usqu'à ce que nous arrivons au dernier pixel du fichier qui se trouve en haut à droite.

Notre Première image va donc resembler à ça:

EXERCICE A2:

```
0000:0000 42 4D 4A 00  00 00 00 00  00 00 1A 00  00 00 0C 00
0000:0010 00 00 04 00  04 00 01 00  18 00 FF FF  00 FF 00 FF
0000:0020 E8 9D 0F FF  FF FF 00 00  FF 00 00 FF  00 00 FF FF
0000:0030 00 00 00 00  FF 00 00 FF  00 00 FF 00  00 FF 00 00
0000:0040 FF 00 FF 00  00 00 FF 00  00 FF
```

L'en-tête du fichier est la même que l'exercice 1, il n'y a que la taille du fichier qui change. Nous allons donc détailler directement le contenu.

Le premier pixel de notre fichier commence à l'adresse 0x1A et il représente le pixel en bas à gauche, ce pixel est codé sur 3 octets. un octet pour la couleur bleu allant de 0 à 255, un octet pour vert allant de 0 à 255 et un octet pour le rouge allant de 0 à 255 (en tenant compte de l'endian si nous l'on n'en tient pas compte l'ordre n'est pas inverser et donc nous obtenons du Rouge puis du Vert puis du bleu).

Sachant que ce pixel doit être de couleur cyan nous auront FF FF 00(qui represente le cyan et qui est codé en hexadecimal le code rvb du cyan est donné dans le site sur le pdf de la SAE) (FF=255). Dans le fichier okteta il ressemble à sa:

```
FF FF 00
```

Le second pixel est de couleur magenta donc nous aurons FF 00 FF (ce pixel est codé de la même manière que le premier, nous remplaçons seulement le code rvb). Dans le fichier okteta il ressemble à sa:

```
FF 00 FF
```

Nous allons continuer à coder comme sa en donnant le code rvb du pixel que l'on veut cela jusqu'a arriver au dernier pixel du fichier qui se trouve en haut à droite.

Dans ce fichier il y a les couleurs (je vais donné les code rvb et l'adresse ou il aparaissent): cyan : FF FF 00 (adresse: 0x1A), magenta: FF 00 FF (adresse: 0x1D), bleu céruléen: E8 9D 0F (adresse: 0x20), bleu: FF 00 00 (adresse: 0x2F), vert: 00 FF 00 (adresse: 0x41),
blanc: FF FF FF (adresse: 0x23), rouge: 00 00 FF (Tout les autres pixels du fichiers).

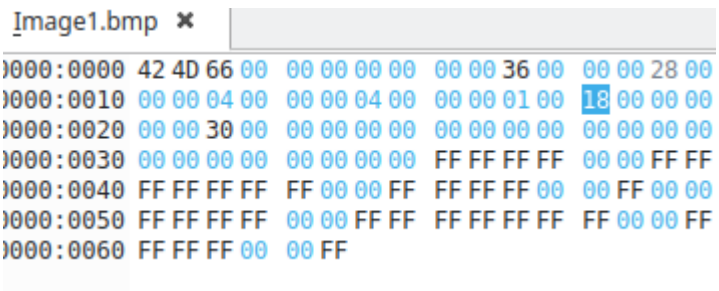
Notre Image va donc resembler à ça:

A partir de maintenant tout les fichier sont en little endian

EXERCICE A3:

Commande à utiliser pour la suite de notre exercice et qui va nous donner une nouvelle façon de coder: convert Image0.bmp bmp3:Image1.bmp

Voici le fichier vu sous okteta:



0. Nous sommes passé d'un fichier de 74 octets à 102. Le calcule est fait de cette manière, nous faisons 74-12 (on enléve le poid du codage du BITMAPCOHEADER) donc 62 + 40 = 102 (40 est le poid du

codage en BITMAPINFOHEADER)

```
18 00
```

1. A l'adresse 0x1C nous pouvons voir le nombre de bits utilisé pour coder un pixel. On peut voir que nous utilisons 24 bits par pixels(00 18 = 8 + 1 * 16=24). Cela correspond à 3 octets.
2. Pour calculer la taille des donnés pixels il faut faire, le nombre de pixels * le nombre d'octet par pixel. Dans notre fichier nous avons 4 pixels par ligne et 4 colonnes donc 4 * 4 = 16 et nous utilisont 3 octets par pixels. Donc 16 * 3 = 48. Nous avons donc 48 octets de donnés pixels. Nous pouvons aussi le voir à l'adresse 0x22 sur 4 octets. Il vaut 00 00 00 30 soit 3 * 16 = 48 octets de donnés pixels.
3. Il n'y a pas de compressions utilisé car lorsque nous regardons l'adresse 0x1E nous pouvons voir que sur 4 bits nous avons que des 0.Ce qui nous indique bien que l'image n'a pas subit une compression.
4. Les pixels sont codés sur 3 octets. Donc il n'y a pas de changements dans le codage des pixels.

EXERCICE A4:

Image2.bmp

```
0000:0000 42 4D 4E 00  00 00 00 00  00 00 3E 00  00 00 28 00
0000:0010 00 00 04 00  00 00 04 00  00 00 01 00  01 00 00 00
0000:0020 00 00 10 00  00 00 00 00  00 00 00 00  00 00 02 00
0000:0030 00 00 02 00  00 00 00 00  FF 00 FF FF  FF 00 50 00
0000:0040 00 00 A0 00  00 00 50 00  00 00 A0 00  00 00
```

```
01 00 (
```

1. A l'adresse 0x1C sur 2 octets nous pouvons voir les valeurs 00 01. Cela nous indique qu'il y'a 1 bits utilisé par pixel.

```
10 00  00 00
```

2. La taille des donnés pixels est de 16 octets. Nous pouvons le voir à l'adresse 0x22 sur 4 octets.
3. Il n'y a pas de compressions utilisé car lorsque nous regardons l'adresse 0x1E nous pouvons voir que sur 4 octets nous avons que des 0. Donc l'image n'a pas subit une compression.
4. Les couleurs de la palettes sont codés sur 4 octets. En effet le premier octet correspond à la couleur bleu. Le deuxième à la couleur vert et le troisième à la couleurs rouge (en tenant compte de l'endiannes). Le dérnier octet est un octet réserver.
5. A l'adresse 0x2E est indiqueé sur 4 octets combien de couleur la palette contient. Ici elle vaut 00 00 00 02. Donc dans cette palette nous avons deux couleurs.
6. Oui le codage des pixels à changé ils sont maintenant codés sur 1 bit.
7.

Image2.bmp ✖

```
0000:0000 42 4D 4E 00  00 00 00 00  00 00 3E 00  00 00 28 00
0000:0010 00 00 04 00  00 00 04 00  00 00 01 00  01 00 00 00
0000:0020 00 00 10 00  00 00 00 00  00 00 00 00  00 00 02 00
0000:0030 00 00 02 00  00 00 00 00  FF 00 FF FF  FF 00 50 00
0000:0040 00 00 A0 00  00 00 50 00  00 00 A0 00  00 00
```

Ceci est le fichier que l'on a ouvert avec okteta avant tout changement. Dans ce fichier vous pouvez voir à l'adresse 0x36 les valeurs 00 00 00 FF(sur 4 octets) qui coresspondent à la couleur rouge (A cette adresse nous sommes dans la palette de couleur. Donc nous allons changer la couleur dans la palette). Donc il suffit de mettre à cette adresse les valeurs FF 00 00 00 pour obtenir du bleu. Voici le nouveau fichier vu sous okteta:

```
0000:0000 42 4D 4E 00  00 00 00 00  00 00 3E 00  00 00 28 00
0000:0010 00 00 04 00  00 00 04 00  00 00 01 00  01 00 00 00
0000:0020 00 00 10 00  00 00 00 00  00 00 00 00  00 00 02 00
0000:0030 00 00 02 00  00 00 FF 00  00 00 FF FF  FF 00 50 00
0000:0040 00 00 A0 00  00 00 50 00  00 00 A0 00  00 00
```

ET voici le resultat:

8.

```
0000:0000 42 4D 4E 00  00 00 00 00  00 00 3E 00  00 00 28 00
0000:0010 00 00 04 00  00 00 04 00  00 00 01 00  01 00 00 00
0000:0020 00 00 10 00  00 00 00 00  00 00 00 00  00 00 02 00
0000:0030 00 00 02 00  00 00 FF 00  00 00 FF FF  FF 00 A0 00
0000:0040 00 00 50 00  00 00 A0 00  00 00 50 00  00 00
```

Voici le fichier vu sous okteta. Nous utilisons les mêmes couleurs que pour la question précedente donc nous avons pas besoin de changer les couleurs de la palette. Or nous allons inverser l'odre des couleurs, pour ce faire tout les 4 octets les valeurs 50 00 00 00 deviendront A0 00 00 00 et les valeurs A0 00 00 00 deviendront 50 00 00 00. Voici la difference:

```
50 00  00 00 A0 00  00 00
```

Ceci correspond au 8 derniers octets de l'image precedente

```
A0 00  00 00 50 00  00 00
```
Ceci correspond au 8 derniers octets de l'image actuelle.

Vous pouvez donc bien voir que nous inversons l'ordre tout les 4 octets pour obtenir l'image demandé.

9.

```
0000:0000 42 4D 4E 00  00 00 00 00  00 00 3E 00  00 00 28 00
0000:0010 00 00 04 00  00 00 04 00  00 00 01 00  01 00 00 00
0000:0020 00 00 10 00  00 00 00 00  00 00 00 00  00 00 02 00
0000:0030 00 00 02 00  00 00 00 00  FF 00 FF FF  FF 00 00 00
0000:0040 00 00 00 00  00 00 00 00  00 00 A0 00  00 00
```

Voici notre image vu sous l'editeur okteta. ET maintenant voici notre image vu avec Imagemagick:

10.

Voici notre logo ouvert avec okteta aprés être passé en mode index de couleurs:

```
0000:0000 42 4D B6 13  02 00 00 00  00 00 76 00  00 00 28 00
0000:0010 00 00 80 02  00 00 A9 01  00 00 01 00  04 00 00 00
0000:0020 00 00 40 13  02 00 00 00  00 00 00 00  00 00 10 00
0000:0030 00 00 10 00  00 00 74 2B  06 00 5C 23  0A 00 70 50
0000:0040 38 00 0C 66  FA 00 20 67  E8 00 51 6F  AB 00 86 35
0000:0050 0C 00 A1 5B  15 00 CD 79  0F 00 9C 6F  52 00 CD 92
0000:0060 3D 00 F4 E8  D8 00 FE FE  FD 00 E5 DF  DC 00 DD C1
0000:0070 A2 00 5C 91  DD 00 CC CC  CC CC CC CC  CC CC CC CC
```

11. On peut le trouver à l'adresse 0x2E sur 4 octets donc il vaut 00 00 00 10 (en little endian). Donc dans cette palette nous pouvons trouver 16 couleurs.(1*16=16)

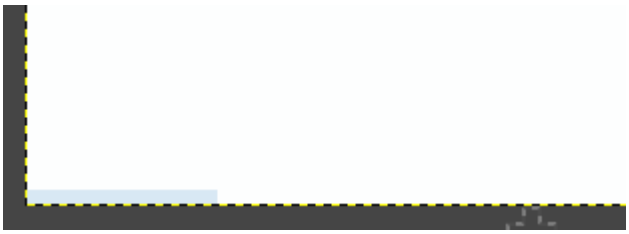
12. Il se trouvent à l'adresse 0x66 sur 4 octets. Il est codés sur 3 octets donc il vaut FE FE FD avec l'octet reserve qui vaut 00.

13. Le tableau de pixel commence à l'adresse 0x76. Cette adresse est donné à l'adresse 0x0A sur 4 octets.

14.

```
0000:0000 42 4D B6 13  02 00 00 00  00 00 76 00  00 00 28 00
0000:0010 00 00 80 02  00 00 A9 01  00 00 01 00  04 00 00 00
0000:0020 00 00 40 13  02 00 00 00  00 00 00 00  00 00 10 00
0000:0030 00 00 10 00  00 00 74 2B  06 00 5C 23  0A 00 70 50
0000:0040 38 00 0C 66  FA 00 20 67  E8 00 51 6F  AB 00 86 35
0000:0050 0C 00 A1 5B  15 00 CD 79  0F 00 9C 6F  52 00 CD 92
0000:0060 3D 00 F4 E8  D8 00 FE FE  FD 00 E5 DF  DC 00 DD C1
0000:0070 A2 00 5C 91  DD 00 BB BB  BB BB BB BB  CC CC CC CC
0000:0080 CC CC CC CC  CC CC CC CC  CC CC CC CC  CC CC CC CC
0000:0090 CC CC CC CC  CC CC CC CC  CC CC CC CC  CC CC CC CC
0000:00A0 CC CC CC CC  CC CC CC CC  CC CC CC CC  CC CC CC CC
0000:00B0 CC CC CC CC  CC CC CC CC  CC CC CC CC  CC CC CC CC
```

Voici le fichier vu sous okteta. Nous pouvons voir qu'à l'adresse 0x76 et jusqu'à l'adresse 0x7B je place des B pour mettre du bleu cela correspond à l'index 11 dasn la palette de couleur. Voici la ligne de bleu que cela me fait:



Et en un peu plus grand:

15. Lorsque l'on diminue le nombre de couleurs dans la palette l'image est plus pixelisé.

Voici se à quoi elle ressemble:

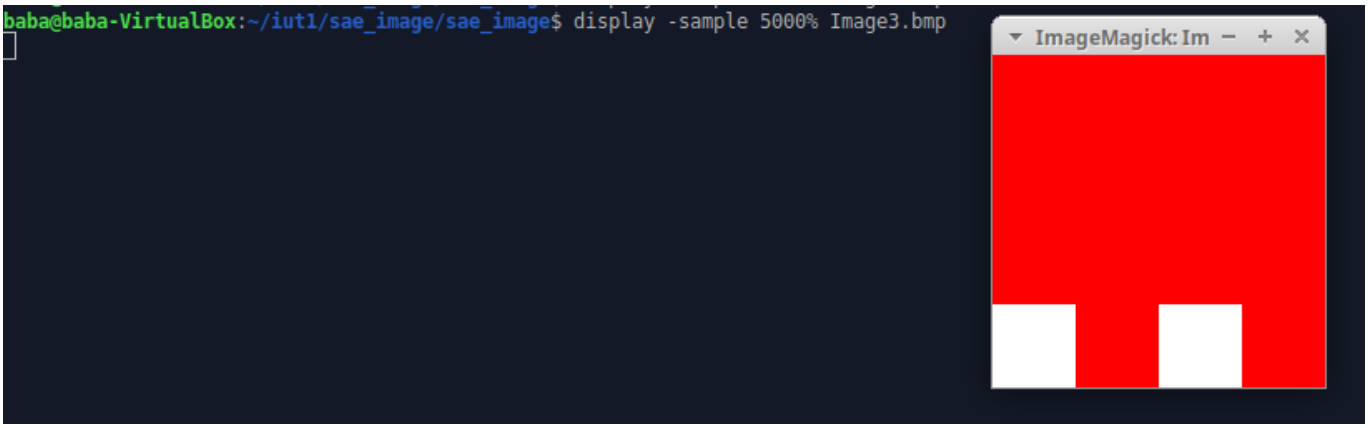
Voici le fichier vu sous okteta:

```
0000:0000 42 4D B6 13  02 00 00 00  00 00 76 00  00 00 28 00
0000:0010 00 00 80 02  00 00 A9 01  00 00 01 00  04 00 00 00
0000:0020 00 00 40 13  02 00 00 00  00 00 00 00  00 00 10 00
0000:0030 00 00 10 00  00 00 A2 59  17 00 FC FB  F9 00 DD C1
0000:0040 A2 00 6A 54  5A 00 00 00  00 00 00 00  00 00 00 00
0000:0050 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
0000:0060 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
0000:0070 00 00 00 00  00 00 11 11  11 11 11 11  11 11 11 11
0000:0080 11 11 11 11  11 11 11 11  11 11 11 11  11 11 11 11
0000:0090 11 11 11 11  11 11 11 11  11 11 11 11  11 11 11 11
0000:00A0 11 11 11 11  11 11 11 11  11 11 11 11  11 11 11 11
0000:00B0 11 11 11 11  11 11 11 11  11 11 11 11  11 11 11 11
0000:00C0 11 11 11 11  11 11 11 11  11 11 11 11  11 11 11 11
0000:00D0 11 11 11 11  11 11 11 11  11 11 11 11  11 11 11 11
```

Nous pouvons voir que le codage des pixels à changé. Nous codons sur 4 bits (obtenu à l'adresse 0x1C). Les valeurs du codage des pixels à aussi changé.

EXERCICE A5)

2. Si nous passons la valeur de la hauteur de l'image à une valeur négatif ce-dernier inversera l'image voici notre imgage de départ:



Pour ce faire nous codons la, valeur -4 en complément à 2 (le C2) et nous allons changer la valeur dans le fichier okteta. -4 vaut FC FF FF FF en C2. En effet 00 00 00 04 (valeur de la hauteur) vaut en binaire 0000 0000 0000 0000 0000 0000 0000 0100. Pour coder cela en C2 il faut inverser les bits à partir du premier un (le 0 devient un 1 et le 1 devient un 0). -4 vaut en C2: 1111 1111 1111 1111 1111 1111 1111 1100 = FF FF FF FC. Sachant que nous sommes en little endian, on doit inverser et on obtient: FC FF FF FF.

La valeur entouré représente le -4. Au depart elle valait 04 00 00 00 (pour la valeur 4).

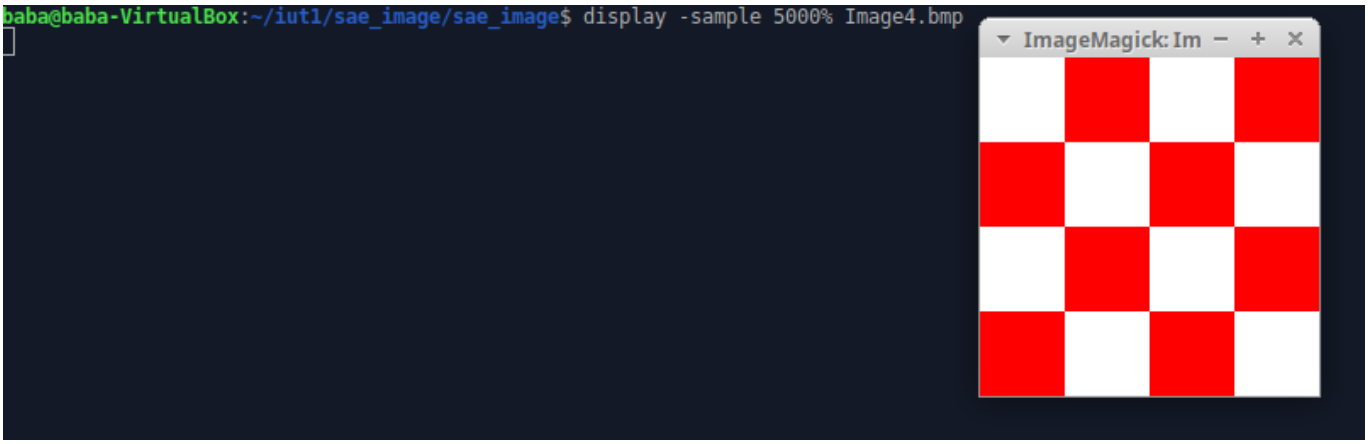
3. Lorsque nous avons finis de mettre la valeur de la hauteur en négatif nous obtenons:

Voici le changement sur le fichier okteta (nous avons passer la hauteur de l'image en négatif, en le codant en C2). La valeur de la hauteur est de 57 FE FF FF. Au depart elle valait 00 00 01 A9 en binaire 0000 0000 0000 0000 0000 0001 1010 1001. On inverse à partir du premier 1 et on obtient : 1111 1111 1111 1111 1111 1110 0101 0111 qui vaut FF FF FE 57.

```
0000:0000 42 4D B6 13  02 00 00 00  00 00 76 00  00 00 28 00
0000:0010 00 00 80 02  00 00 57 FE  FF FF 01 00  04 00 00 00
```

EXERCICE A6)

Voici l'image etudier:



Et voici la taille de l'image:

```
1120 déc.  27 17:49 Image4.bmp
```

1. Le nouveaux fichier vaut 1120 octets. Il y a une palette de couleur qui contient 16^3 couleurs. C'est pourquoi la taille de notre image a augmenter. De plus nous codons les pixels un par un (l'explication est faite dans la question 3) ce type de codage n'est pas adapté au fichier ce qui augmente aussi sa taille.

2. Il est donnée à l'adresse 0x0A est il renvoie à l'adresse 0x436 ou les couleur de l'image débute. L'adresse du debut des pixels est trés loin car il y a beaucoup de couleur dans la palette.

3. Les pixels sont codés sur deux octets le premier octet correspond au nombre de pixel que l'on va colorier et le second octet correspond à la couleur dans la palette.

```
01 00
```

Le premier octet (il se trouve à l'adresse 0x436) vaut 01 cela veut dire que l'on va colorier 1 pixel, le second octet vaut 00 cela correspond à la couleur rouge dans notre palette. Donc nous avons notre premier pixel de couleur rouge.

```
01 01
```

Les deux octet suivants vaut 01 01 cela veut dire que l'on va avoir un pixel de couleur blanc (01 coresspond a la couleur blanche).

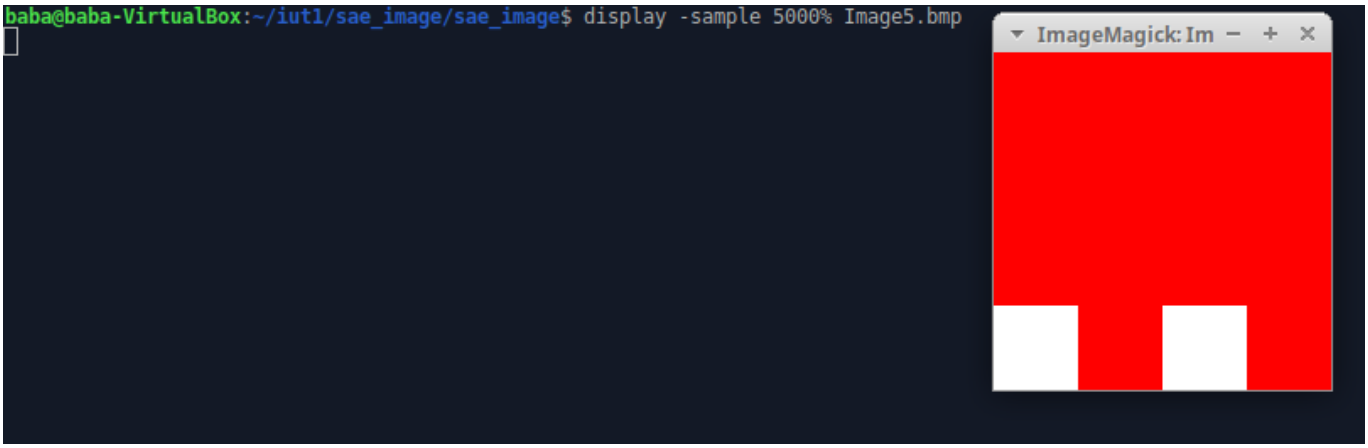
Une fois que nous avons codés notre première ligne nous devont faire un saut, pour ce faire nous entrons sur deux octet 00 00. De cette maniére:

```
01 01 00 00
```

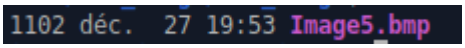
Et pour finir le fichier nous entrerons les valeurs 00 01 pour signaler que c'est la fin du bitmap (du fichier).

A7)

Voici l'image étudié:



Et voici la taille du fichier:



1. L'image 5 vaut 1102 octets. Dans l'image 5 la première ligne est codé de la même façon que l'image 4, or les 3 autres lignes sont codé différament. En effet sur ces 3 lignes nous avons que des pixels rouge donc nous indiquons sur le premier octet le nombre de pixels qu'il y a sur la ligne et sur la deuxième sa couleur. Donc nous gagnons énormement de place car nous codons pas tout les pixels une par une (comme c'est le cas dans l'image 4). C'est pour cela que l'image 5 est moin grand que l'image 4.
2.

Les pixels sont codés sur deux octets le premier octet correspond au nombre de pixel que l'on va colorier et le second correspond à la couleur dans la palette.

```
01 01
```

Le premier octet du fichier se trouve à l'adresse 0x436 et elle vaut 01 cela veut dire que l'on va colorier 1 pixel, le second octet vaut 01 cela correspond à la couleur blanche. Donc nous avons notre premier pixel de couleur blanche.

```
01 00
```

Les deux octets suivants vaut 01 00 cela veut dire que l'on va avoir un pixel de couleur rouge (00 coresspond a la couleur rouge).

```
01 01 00 00
```

Une fois que nous avons codés notre première ligne nous devont faire un saut, pour ce faire nous entrons sur deux octets les valeurs 00 00.

```
04 00 00 00
```

Ensuite pour les 3 autre lignes ont indique que l'on va colorier 4 pixels de couleur rouge et faire un saut de ligne. Donc nous entrons 04 00 00 00 (04 00 veut dire 4 pixels de couleur rouge et 00 00 veut dire le saut de ligne).

EXERCICE A8)

```
00 00 00 00  00 00 02 01  01 00 01 01  00 00 04 00
00 00 04 00  00 00 01 01  01 00 01 01  01 00 00 00
00 01
```

Les 6 premiers octet de l'image correspond à la palette de couleur (00 00 00 00 00 00).

Ensuite nous pouvons voir que nous commençons par mettre deux pixels blanc avec (02 01), un pixel rouge (01 00) et un autre pixel blanc (00 01). Puis nous faisons un saut de ligne (00 00). Nous mettons la ligne en rouge (04 00), en saute une ligne (00 00). Nous remettons la ligne en rouge (04 00), en saute une ligne (00 00). Et sur la dernière ligne on met un pixel blanc (01 01) puis un pixel rouge(01 00), un pixel blanc (01 01) et un dernier pixel rouge (01 00) et nous fermons le fichier avec (00 00 00 01).

Et nous obtenons:

EXERCICE A9)

Pour commencer nous allons rajouter des couleurs dans notre palette.

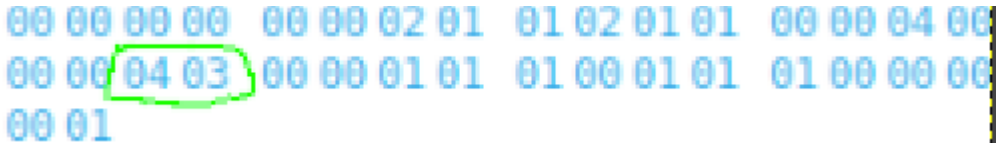
```
00 00 00 01  00 00 00 00  FF 00 FF FF  FF 00 FF 00
00 00 00 FF  00 00 00 00  00 00 00 00  00 00 00 00
```

couleur dans la palette: les couleurs commence à partir du 7éme octet. rouge (00 00 FF) + un octet reservé blanc (FF FF FF) + un octet reservé bleu (FF 00 00) + un octet reservé vert (00 FF 00) + un octet reservé

Voici le codage des pixels vu par okteta:

```
00 00 00 00  00 00 02 01  01 02 01 01  00 00 04 00
00 00 04 03  00 00 01 01  01 00 01 01  01 00 00 00
00 01
```

Le fichier est code de la même maniere que le fichier 8 sauf que lorsque nous avons une ligne verte nous mettons les valeur 04 03 comme ceci:



et pour les pixel bleu nous mettons 01 02 comme ceci:

```
00 00 00 00  00 00 02 01  01 02 01 01  00 00 04 00
00 00 04 03  00 00 01 01  01 00 01 01  01 00 00 00
00 01
```

Nous obtenons cette image:

EXERCICE A10)

Pour cette image la palette de couleur est la même que pour la question précédente. Or le changement est fait dans le code:

Voici le codage des pixel vu par okteta:

```
00 00 00 00  00 00 02 01  01 02 01 01  00 00 01 00
01 03 02 00  00 00 02 03  01 00 01 03  00 00 01 01
01 00 01 01  01 00 00 00  00 01
```

Petit rappel: Pour mettre la valeur rouge on entre 00 qui prendra sa référence dans la palette. Pour le blanc c'est : 01, le bleu c'est: 02 et le vert c'est: 03.

Les 6 premiers octet de l'image correspond à la palette de couleur (00 00 00 00 00 00). Ensuite nous pouvons voir que nous commençons par mettre deux pixels blanc avec(02 01) un pixel bleu (01 02) et un autre pixel blanc (01 01) puis nous faisons un saut de ligne (00 00) (Fin premiére ligne).

voici cette ligne dans okteta:

```
02 01  01 02 01 01  00 00
```

Nous mettons un pixel en rouge (01 00), un pixel en vert (01 03) et deux pixel rouge (02 00) en saute une ligne (00 00) (Fin deuxième ligne)

Nous metton deux pixel en vert (02 03), un pixel rouge (01 00), un pixel vert (01 03), en saute une ligne (00 00) (fin troisième ligne)

Voici la ligne 3 dans okteta:

```
02 03  01 00 01 03  00 00
```

Et sur la dernière ligne on met un pixel blanc (01 01) puis un pixel rouge(01 00) un pixel blanc (01 01) et un dernier pixel rouge (01 00) et nous fermons le fichier avec (00 00 00 01).

Et nous allons obtenir cette image:

Dans la seconde partie de l'exercice il nous est demandé de supprimé les couleurs inutiles dans la palette de couleur est de l'enregistrer dans un fichier nommé Image 9.bmp Notre image 8 fait 1114 octets, lorsque nous supprimons les couleurs inutiles dans la palette Il ne reste plus que 106 octets.

```
 1114 déc.  27 20:19 Image8.bmp
  106 déc.  27 20:24 Image9.bmp
```

voici à quoi resemble le fichier okteta de l'image 9 avec une palette trés réduite:

```
0000:0000 42 4D 4E 04  00 00 00 00  00 00 46 00  00 00 28 00
0000:0010 00 00 04 00  00 00 04 00  00 00 01 00  08 00 01 00
0000:0020 00 00 18 00  00 00 00 00  00 00 00 00  00 00 04 00
0000:0030 00 00 04 00  00 00 00 00  FF 00 FF FF  FF 00 FF 00
0000:0040 00 00 00 FF  00 00 02 01  01 02 01 01  00 00 01 00
0000:0050 01 03 02 00  00 00 02 03  01 00 01 03  00 00 01 01
0000:0060 01 00 01 01  01 00 00 00  00 01
```

Alors que voici une petite partie du fichier de l'image 8:

```
0000:0000 42 4D 5A 04  00 00 00 00  00 00 36 04  00 00 28 00
0000:0010 00 00 04 00  00 00 04 00  00 00 01 00  08 00 01 00
0000:0020 00 00 18 00  00 00 00 00  00 00 00 00  00 00 00 01
0000:0030 00 00 00 01  00 00 00 00  FF 00 FF FF  FF 00 FF 00
0000:0040 00 00 00 FF  00 00 00 00  00 00 00 00  00 00 00 00
0000:0050 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
0000:0060 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
0000:0070 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
0000:0080 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
0000:0090 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
```

La différence est énorme.

PARTIE B:

EXERCICE B1)

```python
from PIL import Image
import os
nom_fichier = os.path.join(os.path.abspath(os.path.dirname(__file__)), "Imagetest.bmp")
#permet d'ouvrir l'image de n'importe quel repertoire
mon_image=Image.open(nom_fichier)
#i est l'image que l'on veut ouvrir (ici c'est l'image test)

def tourner_image(image,position_sortie):
    """
    Fonction qui va faire la transposer de notre image (la trouner de 90° cela fait la même chose)
    parametre:
        entrer:
            image: ceci est l'image que l'on va utiliser
            position_sortie: l'endroit ou mon image modifié va être enregistrer

    """
    sortie=image.copy()
    #on copie cette image dans une variable nommé sortie
    for y in range(image.size[1]):
        #pour chaque ligne de l'image
        for x in range(image.size[0]):    #pour chaque colonne de l'image
            c= image.getpixel( (x,y))
            # on recuper le code rvb du pixel à la position x,y

            sortie.putpixel((y,x),c)
            #on change le pixel de la position colonne ligne à la position ligne colonne
    sortie.save(position_sortie)

tourner_image(mon_image,"./sae_image/Imageout0.bmp")
```

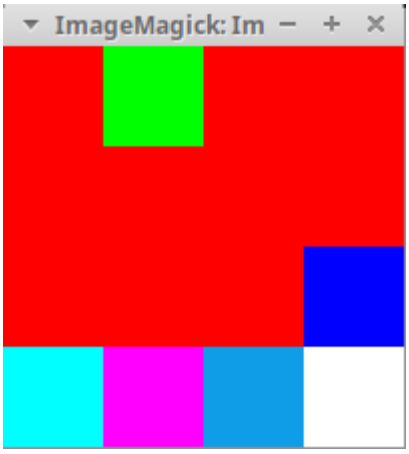
Je commence par ouvrir mon fichier dans une variable nommé mon_image (ligne 3-5) Je crée un fonction pour tourner mon image: Je commence par copier mon image dans une variable nommé sortie (ligne 8)

Puis je parcour les pixels de mon image i (avec mes deux boucle for). (ligne 19-21) Je recupère le code rvb du pixel de mon image i et le stock dans la variable c (c pour couleur) avec c=i.getpixel((x,y)) (ligne 22)

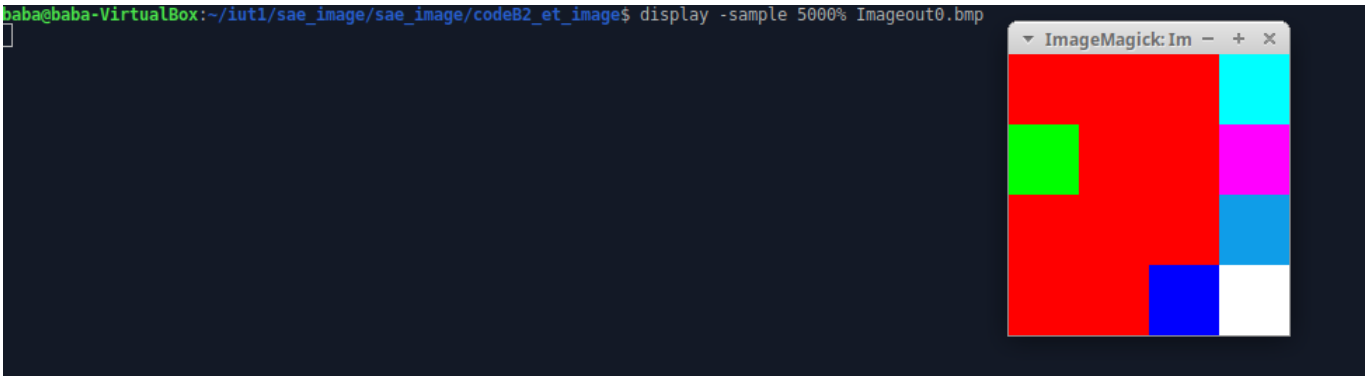
Puis je fait la transposé de mon image, c'est a dire que j'inverse les lignes et les colonnes pour obtenir l'image demandé. Pour ce faire j'utilise: sortie.putpixel((y,x),c) (je change le pixel de la position colonne ligne à la position ligne colonne). Je change donc les pixels de ligne et de colonne de mon image de sortie. (ligne 25)

Puis j'enregistre ce fichier à la position de sortie que l'utilisateur va indiquer (personnelement je l'enregistre dans le repertoire courant codeb2_et_image sous le nom de Imageou0.bmp)

Voici l'image test de départ:



Voici la même image aprés le passage du code:



EXERCICE B2)

```python
from PIL import Image

mon_image=Image.open("./hall-mod_0.bmp")
#i est l'image que l'on veut ouvrir (ici c'est l'image test)

def inverse_image_miroir(image,repertoir_sortie):
    """
    Fonction qui va echanger les pixels de la colonnes du début avec ce de la fin
    Parmaetre:
    entrer:
        image: une image bmp
        repertoire de sortie: l'endroit ou l'image de sortie sera enregistrer.

    """
    sortie=image.copy()   #on copie cette image dans une variable nommé sortie
    width=image.width -1
    #récupere la largeur du fichier (le nombre de pixel que l'on trouve dans le fichier)
    for y in range(image.size[1]):
        #pour chaque ligne de l'image
        for x in range(image.size[0]):    #pour chaque colonne de l'image
            c= image.getpixel( (x,y))
            # on recuper le code rvb du pixel à la position x,y

            sortie.putpixel((width-x,y),c)
            #on change le pixel de le pixel avec le dernier (le premier pixel avec le dernier, le deuxieme avec l'avant dernier...)

    sortie.save(repertoir_sortie)

inverse_image_miroir(mon_image,"./codeB2_et_image/Imageout1.bmp")
```

Dans ce code je commence par ouvrir mon image à la ligne 3.

Pour faire ce qui est demandé je vais crée une fonction qui prend en parametre une image bmp et un chemin pour enregistrer la nouvelle image. Je commence par copier mon image bmp (ligne 15) et j'enregistre la largeur de mon image dans une variable nommé width (ligne 16). Ensuite je parcours mon image par ces lignes et colonnes pour obtenir les pixels. je récupère le code RVB dans la variable c (ligne 21) et je vais changée les pixels de la fin de ma ligne et sur la meme colonne par le pixels de la ligne du début (ligne 24).

Exemple: si mon image fait 10 pixel de largeur je prend le premier pixel et dans mon image copier je change le 10éme pixel par le premier que j'ai récuperer. (le pixel 1 devient le pixel 10 dans mon image copie).

Voici l'image de départ:

Voici l'image obtenu aprés le code:

EXERCICE B3)

```
1  from PIL import Image
2
3  mon_image=Image.open("./IUT-Orleans.bmp")
4  #i est l'image que l'on veut ouvrir (ici c'est le logo de l'iut)
5  def niveau_de_gris(i,repertoire_sortie):
6      """
7      met les pixels de l'imgae dans un niveaux de gris.
8      Parametre:
9          entrer:
10             i: une image bmp
11             repertoire_sortie: le repertoir ou l'on va enregistre l'image de sortie
12     """
13     sortie=i.copy()
14     #on copie cette image dans une variable nommé sortie
15     for y in range(i.size[1]):
16         #pour chaque ligne de l'image
17         for x in range(i.size[0]):    #pour chaque colonne de l'image
18             c= i.getpixel( (x,y))       #on prend le code rvb du pixel à la position x,y
19             nv_col=(c[0]+c[1]+c[2])//3      # on met le pixel dans un niveau de gris
20             sortie.putpixel((x,y),(nv_col,nv_col,nv_col))
21             #on change (le pixel) la couleur du pixel
22
23     sortie.save(repertoire_sortie) #on enregistre cela dans le fichier nommé Imageout2.bmp
24 niveau_de_gris(mon_image,"./codeB2_et_image/Imageout2.bmp")
```

J'ouvre mon image à la ligne 3.

Je vais crée une fonction niveau de gris pour faire ce qui est demandé. A cette fonction nous allons lui attribué deux paramètre celui de l'image dans une varable nommé i et un chemin pour le répèrtoire de sortie.

Dans cette fonction je commence par copié mon image dans une variable nommé sortie. (à la ligne 13) Puis je parcours mon image i et je récupere le code rvb de chacun des pixels dans une variable c.(ligne 17-18)

Pour mettre un pixel en niveau de gris nous allons utilisé la formule donné qui est : (rouge + vert + bleu)/3 Dans la ligne 19 j'utilise cette formule est je le met dans une variable nv_col.

Puis dans le l'image copié je change le code RVB des pixels par le code RVB que j'ai obtenu avec le niveau de gris (ligne 20).

Voici mon image de départ:

Voici l'image aprés le code:

Nous pouvons bien voir que nos pixels sont grisé.

EXERCICE B4)

```python
from PIL import Image

mon_image=Image.open("./IUT-Orleans.bmp")
#i est l'image que l'on veut ouvrir (ici c'est le logo de l'iut)
def noir_blanc(i,rep_sortie):
    """
    met les pixels de l'image en noir et blanc.
    Parametre:
        entrer:
            i: une image bmp
            repertoire_sortie: le repertoir ou l'on va enregistre l'image de sortie
    """
    sortie=i.copy()
    #on copie cette image dans une variable nommé sortie
    for y in range(i.size[1]):
        #pour chaque ligne de l'image
        for x in range(i.size[0]):    #pour chaque colonne de l'image
            c= i.getpixel( (x,y))   #on prend le code rvb du pixel à la position x,y
            nv_col=c[0]**2+c[1]**2+c[2]**2
            if nv_col > 255*255*3/2:          #on verifie si la nouvel couleur du pixel sera noir ou blanc
                sortie.putpixel((x,y),(255,255,255))        # nouvelle couleur (du pixel):couleur blanc
            else:
                sortie.putpixel((x,y),(00,00,00))           # nouvelle couleur (du pixel):couleur noir

    sortie.save(rep_sortie)        #on enregistre cela dans le fichier nommé Imageout3.bmp

noir_blanc(mon_image,"./codeB2_et_image/Imageout3.bmp")
```

J'ouvre l'image demandé (ligne 3).

Pour mettre mon image en noir et blanc je commence par crée une fonction qui prend deux parametre: l'image et le repertoire de sortie.

Par la suite je copie mon image (ligne 13)

Je parcour mon image et je prend le code RVB dans une variable c.

La formule pour mettre le pixel en noir ou en blanc c'est: (Rouge² + Vert² + Bleu²), dans la ligne 19 j'utilise cette formule et le place dans une variable nommé nv_col. Ensuite on vérifie. Si nv_col est supérieur à 255*255*3/2 (cette formule est aussi donné) alors le pixel sera blanc et on va le changer dans l'image copié (ligne 21). Sinon le pixel sera noir et on va le changer (ligne 23).

Ps: si le pixel est blanc son code RVB est: 255, 255, 255 si le pixel est noir son code RVB est: 00, 00, 00.

Voici mon image de départ:

Voici l'image aprés le code:

EXERCICE B5)

Pour cette exercice nous allons diviser les questions en 3 partie, 3 fonctions. Dans la première partie nous allons liberer de la place dans nos pixels pour pouvoir mettre une deuxiéme image. Dans la seconde partie nous allons cacher une image dans une image et dans la troisème partie nous allons retrouver l'image que nous avons cacher.

Partie 1:

```python
from PIL import Image


def cacher(i,b):
    return i-(i%2)+b

def trouver(i):
    return i%2


def libere_place(rep_sortie):
    """
    Fonction qui permet de liberer de la place sur le Rouge de chaque pixel pour liberer un peu de place
    """
    hall=Image.open("./hall-mod_0.bmp")
    sortie=hall.copy()
    for y in range(hall.size[1]):
        #pour chaque ligne de l'image
        for x in range(hall.size[0]):    #pour chaque colonne de l'image
            c= hall.getpixel( (x,y))   #on prend le code rvb du pixel à la position x,y
            valeur_r=c[0]-(c[0]%2)      #on va dégrader la couleur rouge du pixel
            sortie.putpixel((x,y),(valeur_r,c[1],c[2]))
    sortie.save(rep_sortie)        #on enregistre cela dans un fichier


libere_place("./codeB2_et_image/Imageout_steg_0.bmp")
```

Je commence par créé deux fonction qui vont nous permettre de cacher et de retrouver l'image caché (je vais utiliser c'est ces deux fonction dans les deux procaine partie).

Pour liberer de la place je commence par parcourir mon image et de mettre le code RVB des pixels dans une variable c. (ligne 17-20) Ensuite dans valeur_r je soustrait à la valeur rouge (le rouge est la couleur la moin utilisé dans cette image) le reste de la division par deux de cette valeur. (Rouge = Rouge - Rouge%2) Je le fait à la ligne 21. Donc si la couleur rouge avait une valeur paire elle reste le même mais si cette valeur est impair on soustrait 1 à cette valeur pour qu'il soit pair. Cela va nous permettre de cacher et retrouver les pixels de l'image.

J'enregistre ce fichier et je le nomme Imageout_steg_0.bmp

Voici l'image avant le code:

Voici l'image aprés le code libere:

A l'oeil nu l'Humain ne voit aucune différence entre ces deux images.

Partie 2:

Dans cette partie je vait utiliser le code cacher pour cacher une image dans un autre.

La fonction cacher donné pour l'exercice:

```
def cacher(i,b):
    return i-(i%2)+b
```

Et voici ma fonction qui va me permettre de cacher une image:

```
32  def cacher_image(image_princip,image_a_cacher,rep_sortie):
33      """
34      fonction qui va cacher une image dans l'autre
35
36      """
37      sortie=image_princip.copy()
38      for y in range(image_a_cacher.size[1]):
39          #pour chaque ligne de l'image
40          for x in range(image_a_cacher.size[0]):    #pour chaque colonne de l'image
41              c= image_princip.getpixel( (x,y))   #on prend le code rvb du pixel à la position x,y
42              c1=image_a_cacher.getpixel( (x,y))
43              nv_col=c1[0]**2+c1[1]**2+c1[2]**2
44              if nv_col > 255*255*3/2:          #on verifie si la nouvel couleur du pixel sera noir ou blanc
45                  sortie.putpixel((x,y),(cacher(c[0],1),c[1],c[2]))
46              else:
47                  sortie.putpixel((x,y),(cacher(c[0],0),c[1],c[2]))
48      sortie.save(rep_sortie)        #on enregistre cela dans le fichier nommé Imageout3.bmp
49
50
51  image_principal=Image.open("./codeB2_et_image/Imageout_steg_0.bmp")
52  logo_a_cacher=Image.open("./codeB2_et_image/Imageout3.bmp")
53  cacher_image(image_principal,logo_a_cacher,"./codeB2_et_image/Imageout_steg_1.bmp")
```

Dans cette fonction je vais parcourir mon image à cacher que j'ai nommé image_a_cacher. (Je parcoure cette image car elle est plus petite que l'image pricipale, de plus si l'image à cacher était plus grande cela ne fonctionnerait pas.)

Je récupere les pixels de mon image pricipale et de mon image à cacher dans les variables c et c1. (ligne 41-42) Ensuite j'utilise la même formule que l'Exercice 4 pour vérifier si le pixel de l'image à cacher sera noir ou blanc.

Si il est blanc j'utilise la fonction cacher et je le cache avec une valeur de b=1 dans ma fonction caher. (LA couleur rouge du pixel sera un nombre impair cela va nous aider à retrouver notre image dans la partie 3) Si il est noir je le cache avec une valeur de b=0 dans ma fonction caher.(LA couleur rouge du pixel sera un nombre pair cela va nous aider à retrouver notre image dans la partie 3).

Et je finit par le changer dans mon image de sortie. (ligne 45 et 47)

Voici l'image de départ:

Voici l'image de fin:

Vous ne voyez toujours pas de différence à l'oeil nu attendez la paartie 3 pour voir ce qu'il y a dans cette image.

Partie 3:

Dans cette partie je vait utiliser la fonction trouver:

```
7   def trouver(i):
8       return i%2
```

Et je vait utiliser ma fonction retrouver une image cacher:

```
58  def retrouver_image_cacher(image_princip,rep_sortie):
59      """
60      Fonction qui permet de retrouver une image cacher
61      """
62
63      sortie=image_princip.copy()
64      for y in range(image_princip.size[1]):
65          #pour chaque ligne de l'image
66          for x in range(image_princip.size[0]):    #pour chaque colonne de l'image
67              c= image_princip.getpixel( (x,y))   #on prend le code rvb du pixel à la position x,y
68              if trouver(c[0]) == 0:
69                  sortie.putpixel((x,y),(00,00,00))
70              else:
71                  sortie.putpixel((x,y),(255,255,255))
72
73          sortie.save(rep_sortie)        #on enregistre cela dans un fichier
74
75  img_principal_avec_autre_img=Image.open("./codeB2_et_image/Imageout_steg_1.bmp")
76  retrouver_image_cacher(img_principal_avec_autre_img,"./codeB2_et_image/ImageB5_qui_etait_cacher.bmp")
```

Dans cette fonction je parcour mon image et je récupère le code RVB des pixels de l'image. (ligne 64-67)

Par la suite je regarde si la couleur rouge du pixel est pair ou impair. Si il est pair c'est que le pixel est de couleur noir. Alors je vait changer le pixel de l'image de sortie en couleur noir (le code rvb de la couleur noir est: 00 00 00). Si il est impaire c'est que le pixel est de couleur blanche. Alors je vait changer le pixel de l'image de sortie en couleur blanc (code RVB de la couleur blanche est: 255 255 255).

Et grace à ce code je vait obtenir mon image que j'avais cacher au départ.

Voici l'image que nous avons au début:

Et voici l'image qui était cacher à l'interieur:

Maintenant à l'oeil nu nous pouvons voir ce qui était cacher.

Partie B Bonus:

EXERCICE B6)

Dans cette exercice il nous est demandé de caché du texte dans une image. Au début nous commençons par faire les mêmes fonctions que l'exercice B5. Nous allons copier les fonctions cacher, trouver et libere_place. Rapellons-nous que la fonction liberer place permet de degrader le pixel rouge (la couleur la moin utilisé) d'un chiffre si il est impair.

```python
from PIL import Image


def cacher(i,b):
    return i-(i%2)+b

def trouver(i):
    return i%2

mon_image=Image.open("./IUT-Orleans.bmp")
#i est l'image que l'on veut ouvrir (ici c'est le logo de l'iut)

def libere_place(rep_sortie):
    """
    Fonction qui permet de liberer de la place sur le Rouge de chaque pixel pour liberer un peu de place
    """
    hall=Image.open("./hall-mod_0.bmp")
    sortie=hall.copy()
    for y in range(hall.size[1]):
        #pour chaque ligne de l'image
        for x in range(hall.size[0]):    #pour chaque colonne de l'image
            c= hall.getpixel( (x,y))   #on prend le code rvb du pixel à la position x,y
            valeur_r=c[0]-(c[0]%2)      #on va dégrader la couleur rouge du pixel
            sortie.putpixel((x,y),(valeur_r,c[1],c[2]))
    sortie.save(rep_sortie)        #on enregistre cela dans un fichier


libere_place("./codeB2_et_image/Imageout_steg_out0_partie_b6.bmp")
```

J'ai enregistrer le fichier avec la couleur rouge degrader sous le nom de: Image_steg_out0_partie_b6.bmp
J'ai crée une fonction pour cacher mon image. J'ai diviser cette fonction en deux partie:

```python
liste_bin_lettre=[]

for lettre in texte:                #pour chaque lettre dans le mot
    texte = "".join(["{:08b}".format(ord(lettre))])
    #texte = les lettres du texte que nous passons en hexadecimale
    texte = [int(chiffre) for chiffre in texte] #nous convertissont
    for chiffre in texte:
        liste_bin_lettre.append(chiffre)
```

Dans cette partie du code je commence par parcourir mes lettres de mon mot que je vais caché. La ligne 3: "".join("{:08b}".format(ord(lettre))). La fonction ord permet de mettre une lettre en decimal avec la table ASCII. Puis nous allons mettre ce chiffre decimal en binaire sur 1 octet dans une chaine de caractere . Puis avec la ligne 4 je place chaque bit de ma lettre dans la liste. Voici donc un exemple de ce que fait la partie une de ce code.

```
('jeu', [0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1])
```

Nous pouvons voir que le mot jeu à été coder sur 3 octet en binaire.

```
indice=0
sortie=image.copy()
for y in range(image.size[1]):
#pour chaque ligne de l'image
    for x in range(image.size[0]):    #pour chaque colonne de l'image
        c= image.getpixel( (x,y))   #on prend le code rvb du pixel à la position x,y
        if liste_bin_lettre[indice] == 1:
            sortie.putpixel((x,y),(cacher(c[0],1),c[1],c[2]))
        else:
            sortie.putpixel((x,y),(cacher(c[0],0),c[1],c[2]))
        indice+=1

        if indice >= len(liste_bin_lettre):     #verifie que la liste binaire des lettres est fini
            sortie.save(rep_sortie)     #on enregistre cela dans un fichier
            return                          #on sort de la fonction
```

Dans cette partie je parcour mon image avec un degrader de couleur rouge, je recupere le code RVB de chaque pixel dans une variable. Si le chiffre de ma liste binaire vaut 1 alors je le cache dans la couleur rouge de mon image avec un nombre impair. Sinon je le cache avec nombre pair. Puis dans mon dernier if, je verifie que je ne sort pas de la liste pour ne pas savoir d'erreur. Si j'atteint le dernier bit de ma liste alors j'enregistre mon fichier de sortie et je quitte ma fonction. J'enregistre se fichier sous le nom de Image_steg_out1_partie_b6.bmp

Voici mon code en entier:

```
def cacher_texte(image,rep_sortie,texte):
    """
    Fonction qui permet de cacher un texte dans une image.
    entrer:
        image: une image dans lequel nous allons cacher notre texte.
        rep_sortie: un chemin pour la sortie de notre nouveaux fichier
    """
    liste_bin_lettre=[]

    for lettre in texte:            #pour chaque lettre dans le mot
        texte = "".join(["{:08b}".format(ord(lettre))])
        #texte = les lettres du texte que nous passons en hexadecimale via la table ASCII puis que nous convertisson
        texte = [int(chiffre) for chiffre in texte] #nous convertissont en int chaque chiffre codés en str "0000000
        for chiffre in texte:
            liste_bin_lettre.append(chiffre)

    indice=0
    sortie=image.copy()
    for y in range(image.size[1]):
    #pour chaque ligne de l'image
        for x in range(image.size[0]):    #pour chaque colonne de l'image
            c= image.getpixel( (x,y))   #on prend le code rvb du pixel à la position x,y
            if liste_bin_lettre[indice] == 1:
                sortie.putpixel((x,y),(cacher(c[0],1),c[1],c[2]))
            else:
                sortie.putpixel((x,y),(cacher(c[0],0),c[1],c[2]))
            indice+=1

            if indice >= len(liste_bin_lettre):     #verifie que la liste binaire des lettres est fini
                sortie.save(rep_sortie)     #on enregistre cela dans un fichier
                return                          #on sort de la fonction
```

J'appel cette fonction de cette maniere.

```
image_non_code=Image.open("./codeB2_et_image/Imageout_steg_out0_partie_b6.bmp")
cacher_texte(image_non_code,"./codeB2_et_image/Imageout_steg_out1_partie_b6.bmp","Bonus")
```

Voici mon code pour retrouver mon image:

```
72  def retrouver_texte_cacher(image_princip):
73      """
74      Fonction qui permet de retrouver une image cacher
75      """
76      pixel_image=[]
77      cache=""
78      for y in range(image_princip.size[1]):
79          #pour chaque ligne de l'image
80          for x in range(image_princip.size[0]):    #pour chaque colonne de l'image
81              c= image_princip.getpixel( (x,y))   #on prend le code rvb du pixel à l
82              if trouver(c[0]) == 1:      #si la couleur rouge est impaire alors on
83                  pixel_image.append("1")
84              else:
85                  pixel_image.append("0")
86              if len(pixel_image) == 8:       #on verifie si nos chiffre font 1 octe
87                  if pixel_image == ['0', '0', '0', '0', '0', '0', '0', '0']:     #s
88                      break       #alors on arrete le parcours
89                  val=0
90                  for chiffre in range(len(pixel_image)):
91                      puissance=(len(pixel_image)-chiffre-1)
92                      val+=int(pixel_image[chiffre])*(2**puissance)
93                  cache+=chr(val)
94                  pixel_image=[]
95      print(cache)
96
```

Dans cette fonction je commence par parcourir mon image et récuperer le code RVB de chaque pixel dans une variable. Puis je verifie avec la fonction trouver (qui à été donné) si le pixel est impair ou pair (si il vaut 0 ou 1) et je l'ajoute dans une liste. Une fois avoir atteint 1 octet c'est à dire 8 bit, je commence par verfier si ma liste n'est pas une suite de 0 car huit 0 dans une liste veut dire que les lettres de mon mots sont terminé et donc je vait arreter le parcour de pixel et vait afficher le mot. Sinon je parcour ma liste. Pour chaque bit dans ma liste je prend la valeur et le met en puissance de deux pour obtenir mon octet binaire en decimal (ligne 91-92). Exemple si ma liste vaut [0 0 0 0 0 0 1 1] je vait faire 1 + 1*2 qui v a me faire 3.

Une fois avoir eu tout les bits de ma liste je converti mon nombre decimal en une lettre ASCII (grace à la fonction chr). Et j'affiche ce resultat.

Le mot que j'ai caché est: Bonus

Voici l'image avant tout modification:

Voici l'image aprés le degrader de couleur rouge:

Voici l'image dans lequel nous avons caché le texte:

Voici le message que j'ai retrouver (c'est le même que j'ai caché):

Bonus

EXERCICE B7)

Le chiffrement de VERNAM est dit incassable car l'être humain ne peut pas deviner la clef de chiffrement. Si elle essayer toute les possibilités de clefs de chiffrement ils pourrait obtenir des milliers de mots différents et de clef different qui existeront. Or malgrés la difficulté de craqué le code de VERNAM, il existe un seul moyen de le faire. Pour le craqué il faudrait connaitre la clef de chiffrement.

Chiffrons ensemble le mot archiimage avec la clé zskjltubsy qui à été choisis de façon aleatoire et qui est aussis long que le mot à cacher. (Si l'addition des deux lettres ne dépassent pas 26 il n'est pas necessaire de le mettre modulo 26):

a=0 z=25 r=17s=18 c=2 k=10 h=7 j=9 i=8 l=11 i=8 t=19 m=12u=20 a=0 b=1 g=6 s=18 e=4 y=24

a+z = 0+25 = 25 = z
r+s = 17+18 = 35 % 26 = 9 = j c+k = 2+10 = 12 = m h+j = 7+9 = 16 = q i+l = 8+11 = 19 = t i+t = 8+19 = 27% = 1 = b m+u = 12+20 = 32%26 = 6 = g a+b = 0+1 = 1 = b g+s = 6+1 = 7 = y e+y = 4+24 = 28%26 = 2 = c

notre mot chiffré est: zjmqtbgbyc

Exercice B8)

Dans ces conditions que penser de l'usage massif de la surveillance du net dans l'espoir d'empêcher des groupes illégaux de communiquer de façon discrète et secrète?

Sachant que le code de VERNAME est presque impossible à craquer sans la clef de chiffrement si de plus nous mettons ce texte dans une image cela ne fait que renforcer la securite et permet au groupes illégaux de communiquer de façon trés secrète. Donc je pense que la surveillance massif du net n'empechera pas les groupes illégaux de communiquer en toutes discretions.

Pour cette question j'ai repris les codes de la question B6 et j'ai rajouté le code de VERNAME.

Voici mon code de vername:

```
10  def code_vernam(mot_a_cacher):
11      clef=""
12      dico_alph={'a':0,'b':1,'c':2,'d':3,'e':4,'f':5,'g':6,'h':7,'i':8,'j':9,'k':10,'l':11,'m':12,'n':13,'o':14,'p':15,'q':16,'r':17,'s':18,'t':
13      for i in range(len(mot_a_cacher)):          #parcour le mot à cacher
14          lettre_aleatoire=random.choice(string.ascii_letters)       #choisis une lettre de l'alphabet de façon aleatoire
15          clef+=lettre_aleatoire.lower()                             #met cette lettre en petit caractere
16      #j'ai fait les 3 lignes precedente pour avoir une cle de meme taille que mon mot et d'avoir des lettres aleatoires
17      indice=0
18      mot_a_retourner=""
19      for lettre in mot_a_cacher:         #parcours les lettre du mot à cacher
20          if indice == len(clef):
21              indice=0
22          val=dico_alph[lettre]+dico_alph[clef[indice]]
23          val=val%26
24          indice+=1
25          for lettre,valeur in dico_alph.items():
26              if valeur == val:
27                  mot_a_retourner += lettre
28                  break
29      return mot_a_retourner,clef
30
31
```

Je commence par crée une fonction dans laquelle je demande à l'utilisateur d'entrer son mot à cacher. Par la suite je crée un dictionnaire dans lequelle les clés sont les lettres de l'alphabet et les valeurs leur indices (exemple la lettre a à pour indice le 0, le b le 1 ...). Ensuite je parcours mon mot à cacher et pour chaque lettre je demande à l'ordinateur de me donner une lettre de la table ASCII et je met cette lettre en petit car il peut donner aussi des lettres majuscule (ligne 13-15).

Puis je reparcours mon mot (ligne 19). J'additionne l'indice de la lettre du mot avec celle de la cle et je met cette valeur modulo 26 et j'obtient l'indice de ma future lettre(ligne 22-23) . Ensuite je parcour mon dictionnaire de lettre et je prend la lettre qui correspond à l'indice que j'ai calculé juste avant et j'arrete le parcour de mon dictionnaire. Pour finir je retourne mon nouveau mon et ma clef qui va me permettre de retrouver mon mot.

Voici ce que le code de vername donne pour cacher le mot planterunefleur:

```python
print(code_vernam("planterunefleur"))
```

```
PROBLÈMES    SORTIE    CONSOLE DE DÉBOGAGE    TERMINAL    JUPY

baba@baba-VirtualBox:~/iut1/sae_image/sae_image$
baba@baba-VirtualBox:~/iut1/sae_image/sae_image$
baba@baba-VirtualBox:~/iut1/sae_image/sae_image$
baba@baba-VirtualBox:~/iut1/sae_image/sae_image$
baba@baba-VirtualBox:~/iut1/sae_image/sae_image$
baba@baba-VirtualBox:~/iut1/sae_image/sae_image$
baba@baba-VirtualBox:~/iut1/sae_image/sae_image$
baba@baba-VirtualBox:~/iut1/sae_image/sae_image$
baba@baba-VirtualBox:~/iut1/sae_image/sae_image$
baba@baba-VirtualBox:~/iut1/sae_image/sae_image$
('testvnswvagaxfs', 'etsgcjbciwbptlb')
baba@baba-VirtualBox:~/iut1/sae_image/sae_image$
('guecumamecdoxmw', 'rjepbijsryydtsf')
```

Nous pouvons voir que j'ai appelé ma fonction deux fois et que j'ai eu deux mot different car j'utiliser bien des clef aléatoire.

Pour la suite je reutilise les deux codes libere_place et cacher_texte de la question B6: Nous rappelons que le code liberer_place permet de degrader le pixel de couleur rouge (couleur la moin presente) à un chiffre paire en dessous et que le code cacher_texte permet de placer le texte dans une image.

Voici le code liberer_place et celui de cacher_texte:

```python
def libere_place(rep_sortie):
    """
    Fonction qui permet de liberer de la place sur le Rouge de chaque pixel pour liberer un peu de place
    """
    hall=Image.open("./hall-mod_0.bmp")
    sortie=hall.copy()
    for y in range(hall.size[1]):
        #pour chaque ligne de l'image
        for x in range(hall.size[0]):    #pour chaque colonne de l'image
            c= hall.getpixel( (x,y))   #on prend le code rvb du pixel à la position x,y
            valeur_r=c[0]-(c[0]%2)      #on va dégrader la couleur rouge du pixel
            sortie.putpixel((x,y),(valeur_r,c[1],c[2]))
    sortie.save(rep_sortie)        #on enregistre cela dans un fichier


libere_place("./codeB2_et_image/Imageout_steg_out0_partie_b8.bmp")
```

```python
def cacher_texte(image,rep_sortie,texte):
    """
    Fonction qui permet de cacher un texte dans une image.
    entrer:
        image: une image dans lequel nous allons cacher notre texte.
        rep_sortie: un chemin pour la sortie de notre nouveaux fichier
    """
    liste_bin_lettre=[]

    for lettre in texte:            #pour chaque lettre dans le mot
        texte = "".join(["{:08b}".format(ord(lettre))])
        #texte = les lettres du texte que nous passons en hexadecimale via la table ASCII puis que nous
        texte = [int(chiffre) for chiffre in texte] #nous convertissont en int chaque chiffre codés en s
        for chiffre in texte:
            liste_bin_lettre.append(chiffre)
    indice=0
    sortie=image.copy()
    for y in range(image.size[1]):
    #pour chaque ligne de l'image
        for x in range(image.size[0]):    #pour chaque colonne de l'image
            c= image.getpixel( (x,y))   #on prend le code rvb du pixel à la position x,y
            if liste_bin_lettre[indice] == 1:
                sortie.putpixel((x,y),(cacher(c[0],1),c[1],c[2]))
            else:
                sortie.putpixel((x,y),(cacher(c[0],0),c[1],c[2]))
            indice+=1
            if indice >= len(liste_bin_lettre):     #verifie que la liste binaire des lettres est fini
                sortie.save(rep_sortie)     #on enregistre cela dans un fichier
                return                      #on sort de la fonction
```

Maintenant mon mot avec le code de VERNAM est caché dans l'image, comment faire pour le retrouver ?
Pour le retrouver nous allons modifié un tout petit peut le code de retrouver texte de la question B6.

Voici le code:

```python
def retrouver_texte_cacher(image_princip):
    """
    Fonction qui permet de retrouver une image cacher
    """
    pixel_image=[]
    cache=""
    for y in range(image_princip.size[1]):
        #pour chaque ligne de l'image
        for x in range(image_princip.size[0]):    #pour chaque colonne de l'image
            c= image_princip.getpixel( (x,y))   #on prend le code rvb du pixel à
            if trouver(c[0]) == 1:      #si la couleur rouge est impaire alors o
                pixel_image.append("1")
            else:
                pixel_image.append("0")
            if len(pixel_image) == 8:       #on verifie si nos chiffre font 1 oct
                if pixel_image == ['0', '0', '0', '0', '0', '0', '0', '0']:
                    break       #alors on arrete le parcours
                val=0
                for chiffre in range(len(pixel_image)):
                    puissance=(len(pixel_image)-chiffre-1)
                    val+=int(pixel_image[chiffre])*(2**puissance)
                cache+=chr(val)
                pixel_image=[]
    return code_vernam_retrouver(cache,clef)
```

Voici la ligne de code qui change par rapport à la question B6:

```
return code_vernam_retrouver(cache,clef)
```

Nous pouvons voir que dans cette ligne j'appele la fonction qui va nous permettre de retrouver le mot de depart grace à sa clef. Si je n'appele pas cette fonction le code va me retourner le mot qui à été codé avec une clef aleatoire.

Voici le code qui va me permettre donc de retrouver le mot de départ:

```
32  def code_vernam_retrouver(mot,clef):
33      dico_alph={'a':0,'b':1,'c':2,'d':3,'e':4,'f':5,'g':6,'h':7,'i':8,'j':9,'k':1
34      indice=0
35      mot_a_retourner=""
36      for lettre in mot:
37
38          val=dico_alph[lettre]-dico_alph[clef[indice]]
39          val=val%26
40          indice+=1
41          for lettre,valeur in dico_alph.items():
42              if valeur == val:
43                  mot_a_retourner += lettre
44                  break
45      return mot_a_retourner
46
```

Vous pouvez voir que la fonction cacher de vernam et retrouver sont trés similaire. La diffèrence qui nous permet de retrouver le mot de départ est la ligne: 38 car au lieu d'additionner les valeurs je les soustrait pour obtenir la lettre du mot de depart.

Ceci est donc le code de la fonction qui nous permet de retrouver notre texte de départ.

Voici le test que j'ai mis en place pour savoir si mon code fonctionne:

```
124 libere_place("./codeB2_et_image/Imageout_steg_out0_partie_b8.bmp")
125
126 mot_a_cacher=input("veuillez entrer le message que vous vouler cacher: ")
127 mot_a_placer_dans_image,clef=code_vernam(mot_a_cacher)
128 print("le mot que nous avons obtenu apres le passage par le code de vername est: {}, ".format(mot_a_placer_dans_image))
129 print("voici la clef utiliser pour coder ce mot: {}".format(clef))
130
131
132 image_non_code=Image.open("./codeB2_et_image/Imageout_steg_out0_partie_b8.bmp")
133 cacher_texte(image_non_code,"./codeB2_et_image/Imageout_steg_out1_partie_b8.bmp",mot_a_placer_dans_image)
134
135 image_avec_code=Image.open("./codeB2_et_image/Imageout_steg_out1_partie_b8.bmp")
136 mot_depart=retrouver_texte_cacher(image_avec_code)
137 print("Voici le texte que nous avons retrouver dans l'image: {}".format(mot_depart))
138
```

Je commence par appeler ma fonction liberer place (qui degrade la couleur rouge) sur l'image du hall de l'iut et j'enregistre cette nouvelle image sous le nom du fichier imageout_steg0_partie_b8.bmp (ligne 124).

Par la suite je vait demander à l'utilisateur d'entrer le mot qu'il veut cacher. Je vais recuperer le nouveau mot apres le passage du code de VERNAME et la clef qui lui a permit d'être codée. J'affiche le nouveau mot et sa clefde chiffrement (ligne 126-129).

Ensuite j'ouvre l'image ou j'ai degrader la couleur rouge. J'appele ma fonction cacher_texte qui va me permettre de placer le nouveau mot dans l'image et je vais enregistrer cette nouvelle image sous le nom de imageout_steg1_partie_b8.bmp (ligne132-133).

Et pour finir j'appel ma fonction retrouver_texte_cacher qui va me permettre de retrouver mon mot de depart(ligne136). Je fini par afficher le mot de départ avec un petit texte.

Voici le resultat:

```
baba@baba-VirtualBox:~/iut1/sae_image/sae_image$ /bin/python3 /home/baba/iut1/sae_image/sae_image/codeB2_et_image/code_b8.py
veuillez entrer le message que vous vouler cacher: architropbien
le mot que nous avons obtenu apres le passage par le code de vername est: bbjgemjbgffxr,
voici la clef utiliser pour coder ce mot: bkhzwtsnrexte
Voici le texte que nous avons retrouver dans l'image: architropbien
```

Vous pouvez voir que mon mot de depart est: architropbien

Le mot apres le code de VERNAME est : bbjgemjbgffxr (c'est ce mot que nous placons dans l'image)

la clef de chiffrement est bkhzwtsnrexte

le mot retrouver est: architropbien

Et nous retrouvons bien le mot de départ grace à la fonction retouver text image et la fonction retrouver texte VERNAME.

Voici mes images apres leur modification:

Image de départ:

Apres le degrader de couleur:

Image avec le texte:

Nous pouvons voir que cela n'affecte pas nos image.

EXERCICE B9)

L'interet du chiffrement solitaire est de pouvoir communiquer de façon discrète et secrète. Il n'à pas besoin d'ordinateur donc cela le rend encore moin vulnerable au poursuite. De plus cela est faite avec un jeu de carte, même si il donner ce jeu de carte à une personne elle ne comprendrai pas. Cela renforce donc encore plus la securite du chiffrement.