Bit à bit 2

1 Changer un bit particulier

1. Écrire une fonction « met_un » qui prend en entrée un nombre entier n et un nombre entier i, et qui renvoie un nombre entier contenant en binaire partout les mêmes bits que n, sauf éventuellement pour le i-ième bit qui doit être un 1.

   Indication : on pourra entre autres utiliser l'opérateur de décalage vers la gauche <<.
   - l'appel `met_un(104, 2)` devra renvoyer `108`.
   - l'appel `met_un(192 693, 0)` devra renvoyer `192 693`.
2. Écrire de même une fonction « met_zero » qui prend en entrée un nombre entier n et un nombre entier i, et qui renvoie un nombre entier contenant en binaire partout les mêmes bits que n, sauf éventuellement pour le i-ième bit qui doit être un 0.
   - l'appel `met_zero(104, 2)` devra renvoyer `104`.
   - l'appel `met_zero(16, 4)` devra renvoyer `0`.

2 Les opérateurs bit à bit

On souhaite maintenant utiliser un nombre de 32 bits pour stocker des informations de couleurs. On pourrait bien sûr faire plus simplement avec une structure, mais les structures ne sont pas portables dans tous les langages de programmation.

Pour coder une information de couleur sur un nombre de 32 bits, nous allons choisir le standard « ARGB » pour Alpha, Red, Green, Blue. Red indique la teneur en rouge du pixel, Green en vert, Blue en bleu. Et Alpha indique le degré de transparence du pixel. Une valeur de Alpha à 0 donne un pixel transparent, et une valeur de Alpha mise au maximum donne un pixel opaque. Cela permet de créer des effets de fondu pour faire par exemple des assemblages d'images.

Le format classique utilisé par les logiciels de traitement d'image est un format où chaque information est codée sur 8 bits. Nous allons de notre côté réserver, dans cet ordre : 2 bits pour Alpha ; 10 bits pour Red ; 10 bits pour Green ; et 10 bits pour Blue.

Ainsi par ex. :
- 11 0000000000 1111111111 0000000000_2 correspond à un pixel opaque vert.
- 10 1111111111 0000000000 0000000000_2 correspond à un pixel 66,7% opaque rouge.

3. Écrire une fonction « valeur_alpha » qui prend en entrée un nombre entier n sur 32 bits, et qui renvoie le pourcentage d'opacité du pixel correspondant.
   - l'appel `valeur_alpha(3 253 862 398)` devra renvoyer `100%`.

   Indication : on aura besoin cette fois-ci de l'opérateur >> de décalage vers la droite, dont je vous laisse le soin de comprendre l'utilisation.
4. Écrire de même une fonction « valeur_red » qui prend en entrée un nombre entier n sur 32 bits, et qui renvoie le pourcentage de rouge du pixel correspondant.
   - l'appel `valeur_red(3 253 862 398)` devra renvoyer `3%`.
5. Écrire de même une fonction « valeur_green » qui prend en entrée un nombre entier n sur 32 bits, et qui renvoie le pourcentage de vert du pixel correspondant.
   - l'appel `valeur_green(3 253 862 398)` devra renvoyer `12,4%`.
6. Écrire de même une fonction « valeur_blue » qui prend en entrée un nombre entier n sur 32 bits, et qui renvoie le pourcentage de bleu du pixel correspondant.
   - l'appel `valeur_blue(3 253 862 398)` devra renvoyer `99,9%`.
7. Écrire enfin une fonction « pixel » qui prend en entrée trois nombres a, r, g et b entre 0 et 100, et qui renvoie le codage du pixel correspondant à une opacité de a%, une teneur en rouge de r%, une teneur en vert de g% et une teneur en bleu de b%.
   - l'appel `pixel(100, 3, 12.4, 99.9)` devra renvoyer `3 253 862 398`.