LIFAP1 – TD 14 : Structures et Démineur

Objectifs : Approfondir les notions vues dans le TD précédent (structures)
Réviser toutes les notions abordées jusqu'à présent dans les travaux dirigés et travaux pratiques
Au travers d'un exemple de conception, analyser un problème et concevoir sa solution pas à pas

Encore quelques exercices sur les structures images ….

1. Écrire en langage C / C++ un sous-programme permettant d'effectuer la symétrie verticale d'une image (miroir). Le résultat sera stocké dans une nouvelle image qui sera retournée au programme principal.

```
// version en modifiant l'image d'entrée
void miroir (struct image &im)
{
    int i,j;
    int ng;
    for (i=0;i<im.taille_x;i++)
    {
        for (j=0;j<im.taille_y / 2;j++)
        {
            ng = im.tab_img[i][j];
            im.tab_img[i][j]=im.tab_img[i][im.taille_y - j - 1];
            im.tab_img[i][im.taille_y - j - 1] = ng;
        }
    }
}

// version créant une image miroir
void miroir (struct image im, struct image &miroir)
{
    int i,j;

     miroir = im ;

    for (i=0;i<im.taille_x;i++)
    {
        for (j=0;j<im.taille_y;j++)
        {
            miroir.tab_img[i][j]=im.tab_img[i][im.taille_y - j - 1];
        }
    }
}
```

2. Écrire en langage C / C++ un sous-programme permettant de retourner une nouvelle image, calculée comme étant la somme de deux images passées en paramètres. Lorsque la somme des intensités lumineuses des deux pixels ajoutés est supérieure à 255, on la fixera à cette valeur limite.

```
struct image somme (struct image im1, struct image im2)
{
    int i,j;
    struct image somme;
    somme = im1;
    for (i=0;i<im1.taille_x;i++)
    {
        for (j=0;j<im1.taille_y;j++)
        {
            if (somme.tab_img[i][j] + im2.tab_img[i][j] >= 255)
              somme.tab_img[i][j]=255;
```

```
                else
                  somme.tab_img[i][j]+= im2.tab_img[i][j];
              }
        }
        return somme;
}
```

3. Nous souhaitons maintenant construire un dessin animé. Nous allons pour cela gérer plusieurs images. Définir la structure permettant de stocker au plus NB_IMAGE, puis les sous-programmes permettant d'ajouter et de supprimer une image de notre dessin animé.

```
struct dessin_animé
{
      int nb_images;
      struct image T[NB_IMAGE];
};

void ajout_image (struc dessin_anime &DA, struct image im)
{
  DA.T[DA.nb_images] = im ;
  DA.nb_images ++ ;
}

void retire_image (struc dessin_anime &DA, int numero_im)
{
  int i ;
  for (i = numero_im ; i <DA.nb_images -1 ; i++)
  {
    DA.T[i] = DA.T[i+1] ;
  }
  DA.nb_images -- ;
}
```

Pour s'entrainer (après le démineur !)

Écrire en langage C / C++ un sous-programme permettant de proposer le menu suivant

= = MENU= =
0- Saisir une image
1- Afficher intensités min, max et moyenne
2- Seuiller l'image
3- Symétrie de l'image
4- Somme de deux images
5- QUITTER

```
void affiche_menu()
{
  cout<<"LOGICIEL DE TRAITEMENT D'IMAGES"<<endl;
  cout<<"0 - Saisir une image"<<endl;
  cout<<"1 - Extraire des valeurs caractéristiques"<<endl;
  cout<<"2 - Seuiller l'image"<<endl;
  cout<<"3 - Miroir de l'image"<<endl;
  cout<<"4 - Sommer deux images"<<endl;
  cout<<"5 - Quitter"<<endl;
}
```

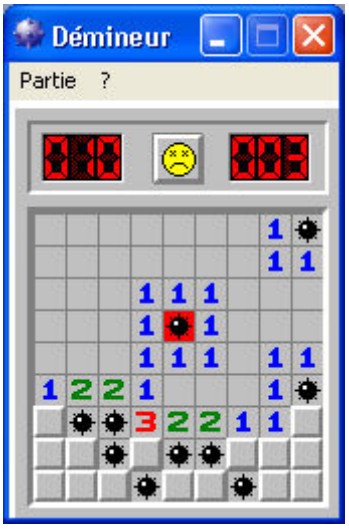
```
void menu ()
{
   int choix;
   struct image ima1,ima2, imas;
   int mini, maxi, moy, seuil;
   do
   {
      affiche_menu();
      cout<<"CHOIX ? : "<<endl;
      cin>>choix;
     switch (choix)
     {
     case 0 :
        ima1=saisir_image();
        afficher_image(ima1);
        break;
     case 1 :
        extraire(ima1,mini,maxi,moy);
        cout<<"min : "<<mini<<"max : "<<maxi<<"moyenne : "<<moy<<endl;
        break;
     case 2 :
        cout<<"quel seuil voulez vous appliquer"<<endl;
        cin>>seuil;
        seuillage(ima1,seuil);
        afficher_image(ima1);
        break;
     case 3 :
        miroir(ima1);
        afficher_image(ima1);
        break;
     case 4 :
        ima2=saisir_image();
        imas=somme(ima1,ima2);
        afficher_image(imas);
        break;

     default : cout<<"au revoir"<<endl;
     }
   }while (choix !=5);
}
```

Le démineur

Il s'agit lors des TD/TP de réaliser un jeu de démineur. Dans le TD, les déclarations de types et les sous-programmes demandés devront être écrits en langage C/C++.

Principe du jeu :



L'objectif est de trouver les mines qui sont cachées aléatoirement par l'ordinateur dans les cases du tableau.

Si la case choisie contient une mine, la partie est perdue. Si la case choisie ne contient pas de mine alors apparaîtra un chiffre indiquant le nombre de mines qui se trouvent dans les 8 cases qui touchent directement la case sélectionnée.

Par exemple si le numéro découvert est un 2, cela indique qu'il y a 2 mines cachées parmi les 8 cases qui touchent directement celle choisie.

Attention : seul un affichage en mode texte sera demandé ici.

Construction du jeu :

1. Structures de données :
   a. Définir 3 constantes *Taille_Grille_X*, *Taille_Grille_Y* et *Mines* permettant de fixer les paramètres généraux du jeu. Dans l'exemple précédent, on aura comme valeurs respectives (9, 9 et 10).

```
#define NB_MINES 10 // définition du nombre de mines à trouver
#define TAILLE_X 9 // définition de la taille de la grille
#define TAILLE_Y 9 // nombre de lignes * nombre de colonnes
```

   b. Définir une structure *case_grille* comportant deux informations :

   - un entier représentant le contenu de cette case (-1 si mine, 0..8 pour le nombre de mines dans les cases adjacentes)
   - un booléen permettant de savoir si cette case a déjà été choisie par l'utilisateur. Cette valeur indiquera si la case doit être dévoilée à l'utilisateur lors de l'affichage.

```
struct case_grille
{
  int valeur;
  bool decouverte;
};
```

   c. Définir la grille de jeu comme étant un tableau 2D de *case_grille*. Pour des besoins ultérieurs, nous allons volontairement surdimensionner la grille de jeu en ajoutant une ligne supplémentaire en haut et en bas et une colonne supplémentaire à droite et à gauche.

```
struct case_grille grille_jeu[TAILLE_X+2][TAILLE_Y+2]
```

2. Initialisation de la grille de jeu : écrire un sous-programme permettant d'initialiser la grille de jeu. Pour chacune des cases de la grille du jeu, la valeur numérique sera initialisée à 0 et le booléen découvert à la valeur "false".

```
void init_grille (struct case_grille grille_jeu[TAILLE_X+2][TAILLE_Y+2])
{
 int i,j;
 for (i=0;i<TAILLE_X+2;i++)
 {
     for (j=0;j<TAILLE_Y+2;j++)
     {
         grille_jeu[i][j].valeur=0;
         grille_jeu[i][j].decouverte=false;
     }
 }
}
```

Pour rappel, un tableau est toujours en donnée/résultat.