(version 12/11/2019)

Notice d'utilisation de la bibliothèque libgraphique

Types

La `libgraphique` définit et utilise deux nouveaux types : `Point` et `Couleur`

Le type `Point`

C'est un type structuré : il est composé de deux **attributs** (ou **champs)** nommés `x` et `y` qui correspondent à des coordonnées (abscisse et ordonnée).
Exemple de déclaration et d'utilisation d'un point.

```
Point p1 ; //declaration d'un Point nomme p1

p1.x = 22 ; //on initialise le champ x de p1 a 22

p1.y =100 ; //on initialise le champ y de p1 a 100

if (p1.x <0) //on peut manipuler les champs comme n'importe quelle variable
   ...
```

Initialisation d'un `Point` (et plus généralement d'une structure)

On peut déclarer et initialiser IMMEDIATEMENT tous les champs d'une structure :

```
Point p1 = {34 , 45} ;
```

...mais **ATTENTION** on ne peut pas tout initialiser d'un coup APRES la déclaration :

```
Point p1 ;
p1 = {34 , 45} ; //CA NE COMPILERA PAS!
```

Vous êtes alors obligé.e d'intialiser chacun des champs un par un :

```
p1.x = 34 ;
p1.y = 45 ;
```

Le type `Couleur`

C'est un type entier qui décrit une couleur en 32 bits. On trouvera dans `libgraphique.h` (et à la fin de ce document) une longue liste de couleurs que l'on peut utiliser directement, dont 16 couleurs standard en français : `bleu, noir, blanc, ....`
On peut fabriquer facilement une nouvelle couleur avec la fonction `fabrique_couleur` (voir ci-dessous).

Fonctions

Gestion de la fenêtre

```
void ouvrir_fenetre(int largeur, int hauteur)
```

Ouvre une fenêtre graphique aux dimensions données, en pixels. Au départ tous les pixels sont noirs.

```
void fermer_fenetre()
```

Termine la session graphique (pas nécessairement le programme principal, attention).

```
void actualiser()
```

Toute modification du type "dessiner" sera invisible tant que la fonction `actualiser` n'a pas été appelée. Ceci permet de faire plusieurs modifications avant de changer l'affichage.

Dessin

Les fonctions de dessin présentées ici utilisent des algorithmes simples, sans *anti-aliasing*.

```
void changer_pixel(Point pix, Couleur couleur)
```

Change la couleur du pixel `pix`, donné par un `Point`, de la couleur `Couleur`.
Exemple :

```
Point p = {50 , 100} ;
changer_pixel(p, bleu) ;
```

```
void dessiner_rectangle(Point coin, int largeur, int hauteur, Couleur couleur)
```

Dessine un rectangle de la couleur donnée ; le point `coin` est le coin supérieur du rectangle, et la largeur et la hauteur correspondent respectivement aux dimensions horizontales et verticales, en pixels.

```
void dessiner_ligne(Point p1, Point p2, Couleur couleur)
```

Dessine une ligne de la couleur donnée entre les deux points.

```
void dessiner_cercle(Point p, int rayon, Couleur couleur)
```

Dessine un cercle de centre `p`, de rayon `rayon`, et de la couleur donnée.

```
void dessiner_disque(Point p, int rayon, Couleur couleur)
```

Dessine un disque de centre `p`, de rayon `rayon`, et de la couleur donnée.

```
void afficher_image(char *nom, Point coin);
```

La chaîne de caractères `nom` doit être le nom d'un fichier bitmap `.bmp` situé dans le même répertoire que le programme. L'image sera affichée selon ses dimensions, le coin supérieur gauche étant donné par le Point `coin`.
IMPORTANT : le fichier BMP peut être encodé en 2, 16, 256 couleurs ou 24 bits **mais ne doit surtout pas être compressé (compression RLE) !**

```
void afficher_image_transparente(char *nom, Point coin, Couleur c);
```

Même effet que la fonction `afficher_image` mais la couleur `c` de l'image `nom` est rendue transparente.

Interaction avec l'utilisateur

`int attendre_touche()`

Cette instruction bloque l'exécution du programme jusqu'à ce que l'utilisateur appuie sur une touche. Le code SDL de la touche est alors renvoyé. Les codes SDL des touches figurent sur

http ://www.libsdl.org/release/SDL-1.2.15/docs/html/sdlkey.html

mais la plupart ont des noms simples commençant par `SDLK_`, comme

`SDLK_A, SDLK_B, SDLK_UP (`*flèche haut*`), SDLK_LEFT, SDLK_SPACE, SDLK_ESCAPE ...`

Exemple :

```
int touche ;
touche = attendre_touche() ;
if (touche == SDLK_DOWN) ...
```

`int attendre_touche_duree(int ms)`

Même effet que la fonction précédente, mais la fonction est bloquante pendant `ms` millisecondes, après quoi elle renvoie 0 si aucune touche n'a été pressée

`Point attendre_clic() ;`

Fonctionne comme `attendre_touche` mais attend un clic de l'utilisateur et renvoie un Point correspondant au pixel cliqué.

Exemple :

```
Point p ;
p = attendre_clic() ;
```

`Point attendre_clic_gauche_droite() ;`

Comme la fonction précédente, mais ajoute un signe négatif devant les coordonnées du point si le clic qui a eu lieu est un clic droit.

`void attente(int ms) ;`

Met le programme en attente durant `ms` millisecondes.

Gestion des couleurs

`Couleur fabrique_couleur(int r, int g, int b);`

Renvoie une Couleur RGB fabriquée d'après les quantités de rouge `r`, vert `g`, et bleu `b` données, qui sont des `int` entre 0 et 255. Par exemple le violet est obtenu en appelant `fabrique_couleur(128,00,128)` ;

`Couleur couleur_point(Point p) ;`

Renvoie la couleur du point `p`. Renvoie `noir` si le point est hors de l'écran.

Tirage aléatoire

`int entier_aleatoire(int n) ;`

Renvoie un entier aléatoire compris entre 0 et n-1 inclus.

Gestion des événements en temps "réel" (optionnel)

Partie optionnelle à ne lire que lorsque vous aurez tout terminé, si vous en avez besoin pour votre projet. Les fonctions comme `attendre_clic` ou `attendre_touche` sont bloquantes : elles bloquent l'exécution du programme en attente d'un clic ou d'une touche. Si l'on désire que le programme continue à s'exécuter tout en surveillant quand même si l'utilisateur clique ou appuie sur des touches, on pourra utiliser les fonctions suivantes. Il s'agit d'une solution sommaire et on peut faire bien plus efficace. L'exemple 5 de la partie 4 utilise ces fonctions sur divers exemples.

```
void reinitialiser_evenements() ;
```

Cette fonction remet à zéro les événements enregistrés.

```
void traiter_evenements() ;
```

Le programme qui gère les graphiques enregistre au fur et à mesure toutes les actions de l'utilisateur (touches et mouvement de souris). Un appel à cette fonction permet de traiter tous les événements n'ayant pas été encore traités. On interrogera ce mécanisme à l'aide des fonctions `touche_a_ete_pressee` et `clic_a_eu_lieu`.

```
int touche_a_ete_pressee(int code) ;
```

Renvoie un booléen indiquant si la touche de code en question (voir `attendre_touche` ci-dessus) a été pressée entre les derniers appels à `reinitialiser_evenements` et `traiter_evenements`.

```
Point clic_a_eu_lieu() ;
```

Renvoie un `Point` dont les coordonnées sont celles du dernier clic entre les derniers appels à `reinitialiser_eveneme` et `traiter_evenements`. Par convention s'il n'y a eu aucun clic, les coordonnées du `Point` renvoyé sont $(-1, -1)$.

```
Point deplacement_souris_a_eu_lieu() ;
```

Renvoie un point de coordonnées relatives souris mesuré entre la dernière réinitialisation et le dernier traitement. Au démarrage, renvoie $(0, 0)$. Si on sort de la fenêtre, renvoie la dernière position reçue.

Affichage de texte

```
void afficher_texte(char *texte, int taille, point coin, Couleur couleur);
```

Affiche le texte `texte` de taille `taille` à la position `coin` (supérieur gauche de la zone de texte) et de couleur `couleur`.

```
Point taille_texte(char *texte, int taille);
```

Renvoie un point qui contient la taille en pixels $(largeur, hauteur)$ de ce texte si on l'affichait. Cette fonction permet de créer un rectangle adapté à la taille d'un texte à afficher.

Comment installer la libgraphique

Si vous souhaitez utiliser la libgraphique sur votre système, vous devez installer les paquets suivants : `libsdl1.2debian, libsdl1.2-dev, libsdl-ttf2.0-0, libsdl-ttf2.0-dev`.

Comment compiler

Si on doit taper la commande en entier, il faut se placer dans le répertoire contenant l'exercice, qui doit être au même niveau que le répertoire `lib` contenant `libgraphique.c`, et taper

```
gcc ../lib/libgraphique.c nom_du_code_source.c -o nom_du_resultat `sdl-config
                    --libs --cflags` -lm -lSDL -lSDL_ttf
```

Attention les ` ne sont pas des apostrophes mais des apostrophes inversées (ALTGR + 7 sur la plupart des claviers). Dans les répertoires préparés pour vous il suffit de taper "make".

Liste des couleurs

En français : argent, blanc, bleu, bleumarine, citronvert, cyan, magenta, gris, jaune, marron, noir, rouge, sarcelle, vert, vertclair, vertolive, violet

En anglais : aliceblue, antiquewhite, aqua, aquamarine, azure, beige, bisque, black, blanchedalmond, blue, blueviolet, brown, burlywood, cadetblue, chartreuse, chocolate, coral, cornflowerblue, cornsilk, crimson, cyan, darkblue, darkcyan, darkgoldenrod, darkgray, darkgreen, darkkhaki, darkmagenta, darkolivegreen, darkorange, darkorchid, darkred, darksalmon, darkseagreen, darkslateblue, darkslategray, darkturquoise, darkviolet, deeppink, deepskyblue, dimgray, dodgerblue, firebrick, floralwhite, forestgreen, fuchsia, gainsboro, ghostwhite, gold, goldenrod, gray, green, greenyellow, honeydew, hotpink, indianred, indigo, ivory, khaki, lavender, lavenderblush, lawngreen, lemonchiffon, lightblue, lightcoral, lightcyan, lightgoldenrodyellow, lightgreen, lightgrey, lightpink, lightsalmon, lightseagreen, lightskyblue, lightslategray, lightsteelblue, lightyellow, lime, limegreen, linen, magenta, maroon, mediumaquamarine, mediumblue, mediumorchid, mediumpurple, mediumseagreen, mediumslateblue, mediumspringgreen, mediumturquoise, mediumvioletred, midnightblue, mintcream, mistyrose, moccasin, navajowhite, navy, oldlace, olive, olivedrab, orange, orangered, orchid, palegoldenrod, palegreen, paleturquoise, palevioletred, papayawhip, peachpuff, peru, pink, plum, powderblue, purple, red, rosybrown, royalblue, saddlebrown, salmon, sandybrown, seagreen, seashell, sienna, silver, skyblue, slateblue, slategray, snow, springgreen, steelblue, tann, teal, thistle, tomato, turquoise, violetlight, wheat, white, whitesmoke, yellow, yellowgreen