Examen Windows API (GI 1)
Polytech EES 4

Copier entièrement le répertoire **ExamAPI_2015** qui se trouve sur la branche de partage dans votre branche d'examen : **\\Groseille\examen\EES16\Exam_API**. Dans le répertoire copié vous trouvez une solution avec les fichiers nécessaires ainsi que la version de démonstration qui montre le comportement final des applications Windows demandées.

Exo1. On souhaite une application Windows qui modélise le comportement d'un digicode. La fenêtre de taille fixe présente en haut une zone d'affichage et un bas un pavé de 12 boutons destinés à taper un code de 4 chiffres.

A. Concevoir une classe **CodeWind** qui modélise la fenêtre de l'application (sans titre, sans bordure, de taille 3x80 par 5x40) avec 12 boutons de taille 80x40 (fenêtres enfants créées dynamiquement dans le constructeur de la classe : attention à la gestion correcte des ressources). Le constructeur prend comme seul paramètre un entier (de quel type ?) entre 0 et 9999 signifiant le code à taper (par défaut **2015**). Faire en sorte que l'on puisse déplacer la fenêtre avec la souris et qu'elle se termine si l'on appuie sur la touche Escape (au point C. il sera nécessaire d'utiliser **SetFocus** pour ramener le focus à la fenêtre mère).

B. La fenêtre démarre avec un fond de couleur vert foncé qui change progressivement (toutes les 2 secondes) vers le rouge foncé (en passant par le jaune). Cela signifie un compte à rebours accordé pour taper le code, dans ce cas 12 secondes, après cela, la fenêtre et l'application se ferment automatiquement. Utiliser un *timer* et pour changer le fond de la fenêtre écrire une courte méthode qui change le pinceau qui lui est associé (accessible avec **GCLP_HBRBACKGROUND** et **SetClassLongPtr**) : créer un nouveau pinceau d'une couleur pré-déclarée dans un tableau (en fonction du temps qui passe) et récupérer l'ancien pinceau qu'il faudra libérer (attention aux *typecast* demandés).

C. L'application se comporte comme un automate à 6 états, en affichant d'une manière **persistante** des messages en haut en fonction de l'état courant : initialement "Taper le code" puis sous forme d'étoiles les caractères correctement tapés (entre 1 et 3). A la fin des 4 chiffres tapés correctement on affiche "GAGNE !!!", on arrête le compte à rebours et on ne modifie plus l'état. Par contre, tout mauvais chiffre ramène l'application dans l'état "Erreur !!!" et l'appui sur '*' ou '#' nous ramène dans l'état initial (le compte à rebours continue).

Exo2. On souhaite une application qui lance un *thread* de manière récursive et que chaque *thread* affiche une fenêtre ressemblant à une figure géométrique simple (cercle, triangle, etc.) de forme, couleur et position aléatoire. Pour limiter le nombre de fenêtre à MAX_WIND (=5) les *threads* de l'application utilisent un sémaphore dont le *handle* est une variable globale.

A. Modifier la classe **FormWind** utilisée en TP pour que le constructeur accepte comme paramètres l'adresse du rectangle englobant, la couleur du fond et la forme (entre 0 et 2). Faire en sorte que la fenêtre soit toujours au 1er plan (quel style ?) et que le click droit la ferme.

B. Déclarer en global le *handle* du sémaphore ainsi qu'une constante MAX_WIND (=5) pour le nombre total de fenêtres. Ecrire un *thread* **ThProc** qui attend un temps infini le sémaphore en question puis crée un nouveau *thread* de lui-même, puis crée (comme variable locale) une fenêtre **FormWind** de forme, couleur et position aléatoire (**rand** ne marche pas bien, utiliser **QueryPerformanceCounter** comme pour **CClsWind**), puis entre dans la boucle de messages et à la fin il relâche (incrémente) le sémaphore de 1.

C. Ecrire une fonction de test et lancement qui crée le sémaphore (attention aux paramètres !), lance un 1er *thread* **ThProc** puis affiche une **MessageBox** avant de quitter. Attention, l'arrêt du *thread* principal (après la fermeture de la **MessageBox**) va mettre fin (brutalement) aux autres *threads*. Tester et comparer à la version de démo.