TP06 – SOUS PROGRAMMES 2

Le but des exercices de cette séance de travaux pratiques est de se familiariser avec le codage en C++ des éléments suivants :

- Définition et appel de sous-programmes

On considère que l'ouverture d'un nouveau projet dans l'environnement de développement Visual Studio Code est acquise. En cas de difficulté avec la gestion d'un projet, reprendre le TP01 à partir de l'étape 3 de la partie 1.

Éteindre correctement le système cible en fin de séance

À partir du terminal, taper la commande suivante :

```
$ sudo poweroff
```

Liste des fonctions à utiliser dans ce TP

- Pour allumer le pixel aux coordonnées (x ,y) de la couleur formée par les valeurs de R, G et B :

```
senseSetRGBpixel(unsigned int x, unsigned int y, uint8_t R, uint8_t G, uint8_t B)
```

- Pour faire une temporisation de *duree* millisecondes

```
sleep_for(milliseconds(duree));  // Bloque l'exécution pendant 'duree' en ms
```

- Pour éteindre toutes les led du senseHat :

```
senseClear()
```

Exercice 1 – Allumer des lignes de led

L'objectif est d'écrire des sous-programmes et un programme principal qui allume une colonne du SenseHat, dont le numéro est tapé par l'utilisateur sur la console.

Préparation (avant la séance de TP)

1. Donnez la vue externe puis l'algorithme (en notation algorithmique) du sous-programme **allumerLigne**, prenant en paramètre formel d'entrée un entier **numLigne**, et permettant d'allumer toutes les leds de la ligne numLigne.
2. Donnez la vue externe puis l'algorithme (en notation algorithmique) du sous-programme **saisirNumLigne**, qui demande à l'utilisateur de saisir un numéro de ligne tant que le numéro tapé n'est pas compris entre 0 et 8, et qui retourne ce numéro.
3. Faire l'algorithme de ce programme.

Réalisation (en séance de TP)

1. Ouvrir une session SSH sur la carte cible. À partir du terminal, lancer la commande : getLab.sh tp6-ex1. Une fois le dossier créé, aller dans le menu Fichier et sélectionner *Ouvrir le dossier*. Choisir le dossier tp6-ex1.
2. Codez en C++ le sous-programme **allumerLigne**, et le programme principal qui appelle ce sous-programme avec un paramètre effectif d'entrée que vous choisirez (un entier). Testez.
3. Codez en C++ le sous-programme **saisirNumLigne**, et le programme principal qui appelle ce sous-programme. Testez.
4. Codez en C++ le programme principal qui appelle en boucle saisirNumLigne et allumerLigne avec l'algorithme suivant :

```
PROGRAMME PRINCIPAL
    //declaration variables
    entier : n
    // actions
    FAIRE
    |     n ← saisirNumLigne()
    |     SI (n ≤ 7) ALORS
    |     |     allumerLigne(n)
    |     SINON
    |     |     senseClear()
    |     FSI
    TQ (n ≠ 8)
FIN PROGRAMME PRINCIPAL
```

5. Testez et faites valider.

Exercice 2 – Allumer le damier

L'objectif est d'allumer toutes les leds avec des couleurs aléatoires.

Préparation (avant la séance de TP)

1. Donnez la vue externe puis l'algorithme (en notation algorithmique) du sous-programme **allumerAleaPixel**, prenant en paramètre formel d'entrée un entier **numLigne**, et un entier **numColonne**, et permettant d'allumer le pixel (numLigne, numColonne), avec une couleur dont les taux de Rouge/Vert/Bleu sont tirés aléatoirement. Pour ce tirage vous utiliserez la fonction `rand()` de la bibliothèque cstdlib (à ajouter dans les #include). Cherchez (sur internet) comment cette fonction peut être utilisée pour tirer un nombre entre 0 et 255.
2. Donnez la vue externe puis l'algorithme (en notation algorithmique) du sous-programme **allumerLigneAlea**, qui est le même sous-programme qu'à l'exercice 1 mais faisant appel à allumeAleaPixel pour allumer les pixels au lieu de `senseSetRGBpixel`.
3. Donnez la vue externe puis l'algorithme (en notation algorithmique) du sous-programme **allumerMatriceAlea** qui allume toutes les lignes en faisant appel à allumerLigneAlea. Ce sous-programme ne prend aucun paramètre formel d'entrée ou de sortie.
4. Faire l'algorithme de ce programme.

Réalisation (en séance de TP)

1. Commencer par copier la partie du code utile du fichier tp6-ex1.cpp. À partir du terminal, lancer la commande : getLab.sh tp6-ex2. Une fois le dossier créé, aller dans le **menu Fichier** et sélectionner **Ouvrir le dossier** et choisir le dossier **tp6-ex2**. Coller le code du presse-papier vers le fichier **tp6-ex2.cpp**
2. Codez en C++ le sous-programme **allumerAleaPixel**, et le programme principal qui appelle ce sous-programme avec deux paramètres effectifs d'entrée que vous choisirez. Testez.
3. Codez en C++ le sous-programme **allumerLigneAlea**, et le programme principal qui appelle ce sous-programme avec une valeur d'entrée que vous choisirez. Testez.
4. Codez en C++ le sous-programme **allumerMatriceAlea**, et le programme principal qui appelle ce sous-programme. Testez
5. Codez en C++ le programme principal qui appelle allumerMatriceAlea un certain nombre de fois (vous choisirez cette valeur), avec une pause (à vous de régler le temps de pause). Testez et faites valider.

Exercice 3 – Utilisation du joystick

L'objectif de cet exercice est de créer un programme qui affiche des caractères sur le senseHat en fonction de la position du Joystick.

Pour utiliser le joystick, il faut déclarer une variable de type `sick_t`, et utiliser la fonction senseWaitForJoystick(). Le résultat retourné permet d'accéder à la position du joystick avec les constantes KEY_UP, KEY_DOWN, KEY_RIGHT, KEY_LEFT, KEY_ENTER, par exemple :

```
int main()
{
	// déclaration variables
	stick_t event;
	int choix;
	//actions
	//…
	event = senseWaitForJoystick();
	choix=event.action;
	if (choix==KEY_DOWN)
		cout << " vous avez tapé DOWN";
	else
		// ...
}
```

Dans cet exercice, vous utiliserez également le sous-programme **`senseShowLetter`**`(char c)`, qui permet d'afficher un caractère sur le senseHat (par exemple `senseShowLetter('A')`).

Préparation (avant la séance de TP)

Pas de préparation

Réalisation (en séance de TP)

1. Donnez la vue externe puis l'algorithme (en notation algorithmique) du sous-programme **afficherDirectionJoystick** qui affiche le caractère N sur le senseHat si le joystick est poussé vers le haut (le nord), S si le joystick est poussé vers le bas (pour Sud), etc. Si l'utilisateur clique sur le joystick (appui central), le sous-programme efface l'affichage sur le senseHat (appel senseClear). Vous pourrez utiliser des branchements conditionnels de type **if** ou **switch**.

2. Donnez la vue externe puis l'algorithme du sous-programme. À partir du terminal, lancer la commande : getLab.sh tp6-ex3. Une fois le dossier créé, aller dans le **menu Fichier** et sélectionner **Ouvrir le dossier** et choisir le dossier **tp6-ex3**.

3. Codez en C++ le sous-programme afficherDirectionJoystick, et le programme principal appelant ce sous-programme.

4. Testez et validez.

5. Donnez la vue externe puis l'algorithme du sous-programme **miseAJourX** qui met à jour l'abscisse x, en incrémentant x de 1 quand l'utilisateur pousse le joystick vers le haut, et décrémente de 1 quand l'utilisateur pousse le joystick vers le bas. Ces modifications ne peuvent se faire que si x reste compris entre 0 et 7. Ce sous-programme retourne la nouvelle valeur de x (en entier). Faire le programme principal qui appelle ce sous-programme, et qui met à jour la valeur de l'abscisse, et si elle a été modifiée qui allume le pixel à la nouvelle coordonnée sur la colonne 0 (en éteignant le pixel précédent).

6. Codez en C++, testez et validez.

7. Le joystick est activé lors de l'appui et du relâchement, vous devriez donc observer un "saut" du pixel de 2 cases au lieu d'une. Pour régler ce problème vous pouvez utiliser event.state (après l'appel de senseWaitForJoystick), qui vaudra KEY_PRESSED, ou KEY_RELEASED. Modifiez votre sous-programme miseAJourX, testez, validez.