Système Embarqué

Journal - TP.04 : Introduction au C

Temps effectué hors des heures de classe

Nous avons travaillé 3h en dehors de la séance de TP.

Synthèse des acquis

Acquis :
- Initialisation des GPIO en C
- Compréhension du fonctionnement du commutateur rotatif
- Travail en groupe à l'aide de Git

Acquis, mais à exercer encore :
- Base du langage C

Questions

Pourrait-on se passer des fichiers d'entête en C ? Si oui, comment ? Si non, pour quelle raison ?

Oui, nous pourrions nous passer des fichiers d'entête mais dans ce cas il serait nécessaire de copier manuellement le contenu du fichier d'entête de chaque inclusion que nous aurions au début de notre fichier de code.

Quelle est l'utilité du pragma #pragma once dans les fichiers d'entête ? Doit-il être accompagné d'une autre directive ? Si oui, laquelle ?

Ce pragma permet de s'assurer que nous n'incluons qu'une seule fois une librairie ou autre partie de notre programme. Il a l'avantage d'éviter un parcours du fichier à inclure s'il reconnaît avoir déjà inclus celle-ci. Cependant, cette instruction est spécifique à notre compilateur. Afin d'obtenir le même résultat indépendamment du compilateur, ou presque puisque l'instruction suivante parcours tout de même le fichier à inclure, il est nécessaire d'écrire le code suivant :

```
#ifndef FILENAME
#define FILENAME
// Some code
#endif
```

Que faut-il placer dans un fichier d'entête ?

Un fichier d'entête contient ce qui sera utiliser par un autre fichier si ce dernier souhaite utiliser certaines fonctions ou variables. Un fichier d'entête contient donc des prototypes de fonctions, des variables et des macros partagées.

Quelle est l'utilité des mots clef extern et static ?

Le mot clé *extern* permet de définir quelles variables et fonctions peuvent être accéder par des fonctions étant dans d'autres fichiers. Par défaut, les variables globales et fonctions qui ne sont pas déclarées *static* sont externes.

Le mot clé *static* permet de rendre une variable globale accessible uniquement dans un fichier, et donc par les fonctions de ce fichier. Contrairement à une variable globale standard qui serait utilisable par n'importe quelles fonctions du projet.
Ce mot clé sert également à rendre une variable statique au sein d'une fonction, c'est à dire que cette fonction ne perdra pas sa valeur à la fin de l'exécution de la fonction.

Comment faut-il procéder pour définir une constante en C ?

Une constante en C peut être défini de deux manières différentes :

```
1 const int dix = 10;
2 // ou
3 #define _Pi 3.1415927
```

Quelle(s) différence(s) existe-t-il entre les instructions #define MAX 10 et const int MAX=10 ; ?

Les deux instructions correspondent à la définition d'une constante, cependant avec la première méthode la constante n'est pas typée. Pour le compilateur, il s'agit de remplacer "textuellement" chaque occurrence de "MAX" par "10".

Comment peut-on définir une énumération en C ? Quelle est son utilité ?

Voici une déclaration d'une énumération :

```
1 enum jour {LUNDI, MARDI, MERCREDI, JEUDI, VENDREDI, SAMEDI, DIMANCHE};
```

Une énumération est une manière spéciale de définir un nouveau type de donnée en utilisant une liste de mot clés. Cela sert à rendre le code plus lisible et compréhensible.

Quelle(s) différence(s) existe-t-il entre une structure en C (struct S) et une classe en Java (class C) ?

Les classes en Java comportent des méthodes alors que les structures en C n'en contiennent pas. Les structures n'ont également pas de constructeurs.

Comment faut-il procéder pour définir un tableau en C ? Peut-on lui donner des valeurs initiales lors de sa définition ?

Un tableau se définit de la manière suivante :

```
1 int tableau[4];
```

Oui nous pouvons le faire, de la façon suivante :

```
1 tableau[4] = {valeur1, valeur2, valeur3, valeur4}
```

Comment faut-il procéder pour obtenir le nombre d'éléments contenus dans un tableau ?

Grâce à la fonction *sizeof(Array)*

Remarques

Il est nécessaire de mettre une légère temporisation entre l'écriture sur chaque 7-segment.

Feedback

Ce TP fut intéressant puisque nous avons pu utiliser plusieurs composants de notre BeagleBone. EN comparaison avec les TPs précédents, il a été plus agréable de coder en C qu'en assembleur. Nous avons également pu nous rendre compte de l'utilité de Git lors d'un travail en groupe.