Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Alan Sueur
Jonathan Rial
Classe I-2a
Systèmes Embarqués
18.11.16

TP04 - Introduction au C

Réponses aux questions

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

Les fichiers d'entête créés par le programmeur ne sont pas obligatoires. Ils servent à déclarer les attributs et les fonctions un peu comme ferait une interface pour une classe en Java.
En C on utilise la méthode *include*"*header.h*" pour inclure le code du fichier d'entête. Cela reviendrait au même si on faisait un copier-coller du code ce qui dégraderait la clarté et la compréhension du code.
Le système utilisera cette syntaxe $\#include < file >$ alors que les fichiers d'entête utiliseront cette syntaxe $\#include"file"$.

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

Si le fichier a déjà été inclut une fois, alors il ne sera pas réinclut une deuxième fois. Le fichier ne peut être compilé qu'une seule fois. Il ne doit pas nécessairement être accompagné d'une autre directive. Une bonne alternative de cette instruction serait :

```
#ifndef AM335X_GPIO_H
#define AM335X_GPIO_H
...
#endif
```

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

Dans un fichier d'entête il faudra placer *pragma once* et $ifndef$ afin que les informations contenues dans ces fichiers ne soient pas dupliquées.

```
#pragma once
#ifndef wheel_h
#define wheel_h
enum ...
extern void ...
#endif
```

Alan Sueur
Jonathan Rial
Classe I-2a
Systèmes Embarqués
18.11.16

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

Le mot-clef *extern* permet de déclarer une variable ou une méthode globale sans la définir. Ce n'est plus obligatoire depuis les dernières versions de C.
Static permet quand à lui :

— De garder sa valeur à travers les différentes invocations.
— De n'être vu que dans le fichier dans lequel il est déclaré.

Par exemple avec ce code :

```
void foo() {
    int a = 10;
    static int sa = 10;
    a += 5;
    sa += 5;
    printf("a = %d, sa = %d\n", a, sa);
}

int main() {}
    int i;
    for (i = 0; i < 3; ++i)
    foo();
}
```

Ceci va ressortir le résultat suivant :

```
// a = 15, sa = 15
// a = 15, sa = 20
// a = 15, sa = 25
```

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

On peut utiliser *const* ou alors $\#define$, dans les deux cas la variable ne pourra être modifiée.

Quelles différences existe-t-il entre les instructions $\#define\ MAX\ 10$ et $const\ int\ MAX = 10$?

Dans les deux cas, on crée une constante. Utiliser *const* est plus restrictif que $\#define$. On ne peut pas utiliser *const* pour les tableau par exemple :

```
static const int c = 10;
static int t[c];    // ko
```

Il faut alors faire :

```
#define c 10
static int t[c];    // ok
```

Avec *const*, on ne peut également pas initialiser des variables *static* et on ne peut pas l'utiliser directement dans un switch.

Alan Sueur
Jonathan Rial
Classe I-2a

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

Une énumération se défini comme suit :

```
enum my_enumeration {..., ...};
```

Par exemple :

```
enum strategy {RANDOM, IMMEDIATE, SEARCH};
enum strategy my_strategy = IMMEDIATE;
```

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

Une structure en C est en quelque sorte un tableau permettant de regrouper des éléments mais en mixant différents types. Par exemple un tableau de int et de String.
Dans une structure, on ne définit pas de méthodes contrairement à une classe.

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

Pour définir un tableau en C il faut utiliser la syntaxe suivante :

```
type arrayName [ arraySize ];
```

Par exemple :

```
double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};
```

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

Il faut diviser par la taille de l'élément pour trouver le nombre d'éléments contenus dans le tableau.

```
int arr[NUM_OF_ELEMS];
size_t NumberOfElements = sizeof(arr)/sizeof(arr[0]);
```

Alan Sueur
Jonathan Rial
Classe I-2a
Systèmes Embarqués
18.11.16

Heures de travail

En dehors des heures de cours, nous avons dû travailler 2h40. Durant le cours nous avons pu réalisé les tâches suivantes :

- Comprendre comment fonctionne le C.
- Rechercher des réponses à nos questions concernant la structure du C.
- Comprendre les étapes que nous avons réalisé en classe avec le prof.
- Continuer le projet et tester si cela fonctionnait.

A la maison, nous avons dû finir les tâches suivantes :

- Répondre aux questions 1-10. [1h20]
- Terminer et tester le projet. [1h00]
- Rédiger ce rapport. [0h20]

Synthèse

Non acquis

- Rien

Partiellement acquis

- Utilisation d'une structure.

Acquis

- Comprendre la syntaxe du C, créer des fonctions, etc.
- Comprendre les nouveaux mots-clés : static, enum, pragma once, ifndef
- Utilisation d'un fichier *.h* pour définir les en-têtes.

Remarques & Feedback

Ce premier TP sur le code C nous a paru bien compliqué comme introduction. Comme nous n'avons pas encore vu la théorie aux cours nous avons dû chercher pas mal d'informations sur internet.
Nous avons tout de même terminé rapidement le tp grâce à l'aide du prof mais nous avons préférer le tp concernant l'introduction à l'assembleur qu'au C.