# Architecture générale et logicielle de systèmes embarqués – TP

Le TP du module Architecture générale et logicielle de systèmes embarqués est un code en langage C à développer.

## Pourquoi le langage C ?

Selon un sondage réalisé en 2019 au sein de la communauté Développez.com, les 5 premiers langages utilisés pour les systèmes embarqués sont le C, C++, Arduino, Python et C#.

Certains langages se veulent dédiés à cet usage parmi lesquels se trouvent Ada. Des langages proches de la machine comme le C et dans une moindre mesure le C++ sont aussi utilisés. Le langage Assembleur reste encore un choix approprié pour les systèmes soumis à des contraintes sévères de temps réel (j'en ai fait l'expérience).

https://embarque.developpez.com/actu/276535/Sondage-quels-langages-utilisez-vous-pour-le-developpement-de-systemes-embarques-en-2019-Et-pourquoi-Partagez-votre-experience/

## Comment développer en C ?

La plupart des IDE avec un compilateur C fait l'affaire.
Les IDE préconisés :

- Visual Studio Code (Je me sers de celui-ci pour mes projets React Native et C)
- Eclipse
- NetBeans
- SublimeText
- Atom
- Code ::Blocks
- CodeLite
- CodeWarrior
- Dev-C++

## Hello World !

```c
#include <stdio.h>
int main() {
  // printf() displays the string inside quotation
  printf("Hello, World!");
  return 0;
}
```

Le langage C fait appel à des fichiers headers (des bibliothèques) qui incluent des méthodes. Ici ***stdio.h*** pour « Standard Input/ Ouptput Header » pour utiliser le ***printf*** (Afficher un message à l'écran. Elle fait partie des bibliothèques standard du C.
https://fr.wikipedia.org/wiki/Biblioth%C3%A8que_standard_du_C

La fonction main, principale donc, sera la première méthode lancée à l'exécution.

## Quel programme va-t-on développer ?

Un thermostat pour une climatisation réversible ! Et oui rien que ça ! enfin presque … aucune climatisation réversible ne sera fournie dans le cadre du TP faute de moyens :-/

Le but de l'exécutable (que l'on imagine intégré à la climatisation réversible en tant que firmware) sera de :

- Définir en entrée la température que l'on souhaite dans une pièce
- Simuler une prise de température de la pièce en créant une méthode qui permette de générer un chiffre aléatoire en 15° et 25°. Cette méthode sera appelée par interruption périodiquement
- A réception de la prise de température, effectuer des traitements sur une variable (un octet) qui permettra le contrôle de la climatisation (Chaud / froid) et 3 puissances selon la différence de température relevée

Ce TP fera appel à différentes notions basiques du langage C :

- Entrées / sorties (saisie clavier, affichage écran)
- Notation hexadécimale
- Interruptions
- Constantes / volatile
- Macros-préprocesseurs
- Génération de nombres aléatoires
- Masquages de bits

Le TP est découpé en 5 étapes. Une étape vaut donc 4 points. 4 points étant un programme qui fonctionne, le code doit aussi être « propre ».

## 1ere étape

### Que faire ?

Créer un programme qui effectue une boucle infinie et appel une méthode « ***OnAlarm*** » toutes les 2 secondes

### De quoi a-t-on besoin ?

- Initialiser une variable de type ***sigaction*** : Cette variable de type structure va paramètre les actions attribuées à un signal. C'est là où on définit l'adresse de la méthode à appeler http://manpagesfr.free.fr/man/man2/sigaction.2.html
- Appeler la méthode ***sigaction*** avec pour paramètres le type de signal et la variable de type ***sigaction***
- Appeler la méthode ***alarm*** http://manpagesfr.free.fr/man/man2/alarm.2.html

### Résultat attendu

```
BEGIN

 ALARM

END
```

## 2ème étape

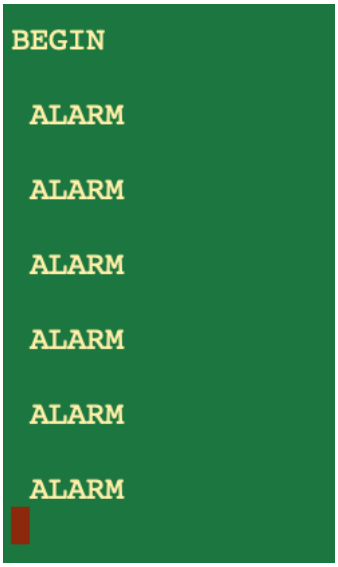
### Que faire ?

Reprendre le code précédent, mais appeler la méthode toutes les 2 secondes indéfiniment.

### De quoi as-t-on besoin ?

Le nombre 2 utilisé doit être défini dans une macro-préprocesseur.
https://www.tutorialspoint.com/cprogramming/c_preprocessors.htm

### Résultat attendu



## 3éme étape

### Que faire ?

Reprendre le code précédent et appeler une méthode ***getTemp*** dans la méthode ***onAlarm***. La méthode ***onAlarm*** affichera le résultat de la méthode ***getTemp***.
La méthode ***getTemp*** générera un nombre aléatoire en 15 et 25.

Les nombres 15 et 25 seront aussi définis dans des macros.

### De quoi as-t-on besoin ?

- Des méthodes :
  - ***Srand***
  - ***Rand*** : http://manpagesfr.free.fr/man/man3/rand.3.html
  - ***Time*** : http://manpagesfr.free.fr/man/man2/time.2.html

### Résultat attendu

```
BEGIN

ALARM
Temp : 20°

ALARM
Temp : 22°

ALARM
Temp : 15°

ALARM
Temp : 17°

ALARM
Temp : 22°

ALARM
Temp : 17°
```

# 4ème étape

## Que faire ?

Reprendre le code précédent et

- ajouter au début du programme avant l'initialisation du timer le fait de pouvoir entrer un chiffre numérique. Pour cela vous avez besoin de la méthode ***scanf*** qui permet de récupérer une entrée clavier. La variable sera de type entier et nommée ***temp_target***
- Faire évoluer la méthode ***onAlarm*** pour détecter si la température entrée est supérieure à la température de la pièce.
- Afficher le mode de la climatisation à effectuer (TEMP_OK, CHAUFFER, REFROIDIR)
- Initialiser une variable qui influencera notre climatiseur selon ce schéma ci-dessous. Par convention on utilise une variable ***unsigned char*** pour stocker un octet.

| BIT_MODE_1 | BIT_MODE_2 | | | | | BIT_POWER_1 | BIT_POWER_2 |
|---|---|---|---|---|---|---|---|
| X | X | 0 | 0 | 0 | 0 | X | X |

Les 2 bits de poids forts sont les bits du mode du climatiseur :

- 00 : En veille
- 01 : Chauffage
- 10 : Froid

Nous devons donc avoir comme valeur à la fin de la méthode OnAlarm pour cette variable :

- En veille : Binaire : 00000000, Décimal : 0, Hexadécimal : 0x0
- Chauffage : Binaire : 01000000, Décimal : 64, Hexadécimal : 0x40
- Froid : Binaire : 10000000, Décimal : Décimal : 128, Hexadécimal : 0x80

Nous utiliserons les 2 bits de poids faible à la prochaine étape. Ils seront utilisés pour setter la puissance.

## De quoi à ton besoin ?

- Méthode ***scanf*** : http://manpagesfr.free.fr/man/man3/scanf.3.html

## Résultat attendu

```
 BEGIN
Enter temp target :18

 ALARM
 Temp : 20°
 REFROIDIR
 RESULT : 128
    128 = 00000000 00000000 00000000 10000000

 ALARM
 Temp : 22°
 REFROIDIR
 RESULT : 128
    128 = 00000000 00000000 00000000 10000000

 ALARM
 Temp : 15°
 CHAUFFER
 RESULT : 64
     64 = 00000000 00000000 00000000 01000000

 ALARM
 Temp : 22°
 REFROIDIR
 RESULT : 128
    128 = 00000000 00000000 00000000 10000000
```

## 5ème étape

### Quoi faire ?

Reprendre le code précédent en ajoutant la notion de puissance sur l'octet suivant la règle suivante :

- Si la différence de température est égale à 1°, setter les bits de poids faibles à 01
- Si la différence de température est égale à 2°, setter les bits de poids faibles à 10
- Si la différence de température est supérieure ou égale à 3°, setter les bits de poids faibles à 11

Exemples :

- Température cible à 20° avec une température de pièce à 18°, l'octet sera égal à 01000010 (Binaire), 66 (Décimal), 0x42 (Hexadécimal)
- Température cible à 20° avec une température de pièce à 25°, l'octet sera égal à 10000011 (Binaire), 131 (Décimal), 0x83 (Hexadécimal)

Il convient bien sûr de modifier l'octet en modifiant les bits de poids fort d'abord (le mode) puis de modifier les bits de poids faibles ensuite (la puissance)

### De quoi as-t-on besoin ?

De technique de masquage de bits.
https://www.learn-c.org/en/Bitmasks
https://people.scs.carleton.ca/~sbtajali/1002/slides/8_Bits.pdf

### Résultat attendu

```
 BEGIN
Enter temp target :18

 ALARM
 Temp : 19°
 REFROIDIR
 RESULT : 129, diff : 1
    129 = 00000000 00000000 00000000 10000001

 ALARM
 Temp : 18°
 TEMP OK
 RESULT : 0, diff : 0
      0 = 00000000 00000000 00000000 00000000

 ALARM
 Temp : 15°
 CHAUFFER
 RESULT : 67, diff : 3
     67 = 00000000 00000000 00000000 01000011

 ALARM
 Temp : 22°
 REFROIDIR
 RESULT : 131, diff : 3
    131 = 00000000 00000000 00000000 10000011
```