Introduction

Le C est le langage le plus fréquemment utilisé pour la programmation des microcontrôleurs. Sa syntaxe ressemble beaucoup à celle de Java.
Ce document n'est pas un cours sur le C, il a pour but de mettre en évidence les principales différences entre C et les langages que vous avez déjà étudiés (Python et Java).
Parmi les différences fondamentales, citons :

- C est un langage compilé, alors que Python est un langage interprété et que JAVA est un mixte des deux puisqu'il est compilé pour une machine virtuelle. Cela rend le programme non portable, ce qui n'est en général pas une contrainte dans le cas des systèmes à microcontrôleur, puisque les programmes sont de toute façon développés pour une plateforme spécifique. Par contre, cela rend l'exécution du programme plus rapide.
- C ne possède pas de gestion dynamique de la mémoire du type *Garbage Collector*. L'utilisation de variables dynamiques est possible, mais doit être entièrement gérée par le programme.
- C n'a pas de gestion des exceptions (comme l'instruction *try* de Python). A nouveau, ces tests doivent être géré par le programme (division par zéro, débordement de variables, …).

Les variables

Les variables en C sont typées, les types sont globalement les mêmes que ceux de Java (cf. guide de programmation).
Les variables doivent être explicitement définies avant d'être utilisées.
Les variables définies en dehors des fonctions sont globales.
Les variables définies dans une fonction sont locales à la fonction.
Si une variable locale porte le même nom qu'une variable globale, elle la "masque", rendant l'accès à la variable globale impossible dans la fonction où elle existe.

Le type booléen

Il n'existe pas de type booléen, les instructions conditionnelles (if, while, switch) ont comme paramètres un entier (ou une expression dont le résultat est un entier), qui est considéré comme faux si il vaut 0 et vrai sinon.
De même, les opérateurs logiques (!, && et ||) et relationnels (==, !=, <, >, <= et >=) ont comme résultat 0 ou 1 pour *vrai* ou *faux*.

C n'est pas un langage orienté objet

Les classes n'existent donc pas en C, de même que tous les concepts associés (polymorphisme, héritage, méthode, encapsulation).
On peut toutefois définir un type de donnée appelé *structure* pour grouper plusieurs données, comme les champs d'une classe. Par exemple, la variable *PORTBbits* (décrite dans le guide de programmation) est une structure.

Les fonctions

En C, toutes les fonctions sont globales (pas de fonction définie au sein d'une autre fonction).
Cela implique que leur nom doit être unique (les noms sont sensibles à la casse).
La notion de surcharge de fonctions n'existe pas.

Modularité

Un programme C peut être découpé en plusieurs fichiers. Cette découpe peut être choisie arbitrairement par le programmeur.
Lors de la compilation du projet, chaque fichier est compilé séparément. Il arrive fréquemment que les variables et fonctions définies dans un fichier soient utilisées dans un autre fichier. Par exemple, prenons deux fichiers *main.c* et *math.c* contenant chacun une fonction :

main.c

```
#include "math.h"

int main() {
   int a = 2 ;
   int b = 3 ;

   c = pow(a,b) ;
         .
         .
         .
}
```

math.c

```
int c;

int pow(int base, int exp) {
   long result = 1;

   while (exp > 0) {
      result *= base;
      exp--
   }
   return(result);
}
```

Pour que le compilateur puisse compiler le fichier *main.c*, il doit avoir connaitre la fonction *pow()*. Toutefois, il n'a pas besoin de la définition complète de la fonction, il lui suffit de connaître ce qu'on appelle le prototype ou la déclaration de la fonction :

```
long pow(int base, int exp);
```

Ce dernier contient toutes les informations dont il a besoin, à savoir le nombre de paramètres et leur type, ainsi que le type retourné par la fonction.
Le même mécanisme s'applique aussi aux variables globales partagées par plusieurs fichiers.
Ces déclarations sont rassemblées dans un fichier "d'en-tête" (Header file en anglais, d'où l'extension .h). Ce fichier doit être inclus dans les fichiers utilisant les fonctions qui y sont déclarées ; dans notre exemple, la première ligne de *main.c* : elle inclut le fichier *math.h*, jouant un rôle similaire à l'instruction *import* de Python et JAVA.
Le fichier *math.h* de notre exemple sera donc :

```
// variable globale définie dans math.c et utilisée dans main.c
extern int c;

// Déclaration de la fonction pow, qui calcule base^exp
int pow(int base, int exp);

// définit la constante pi
#define PI   3.14
```

Pour différencier la déclaration et la définition d'une variable, on ajoute le mot-clé *extern* devant la déclaration, pour préciser que cette variable est définie en dehors du fichier où le fichier d'en-tête est inclus.
La dernière ligne de *math.h* est une macro de pré-compilation (comme *#include*). Elle permet ici de définir la valeur constante π de manière symbolique, sans utiliser d'espace mémoire.
Une macro de pré-compilation est une commande que le compilateur exécute avant la compilation proprement dite. Il s'agit en fait de commandes d'édition du code :

- La macro *#define* indique au compilateur qu'il doit rechercher son 1er argument (PI dans notre exemple) dans le code et le remplacer par son 2ème argument (3.14), exactement comme la commande *remplacer* d'un éditeur de texte.
- De même, la macro *#include* indique qu'il faut la remplacer par le contenu du fichier en paramètre.

Il existe d'autres macros de pré-compilation, mais nous ne les aborderons pas dans ce document.