Formation

Bases du C++ embarqué

Remarques

- je peux faire des erreurs
- les slides et le code associé sont sur github
- si c'est pas clair posez des questions
- je pars du principe que vous savez écrire du C
- tout est testé sur des ST Nucleo L432KC (carte du PAMI)
- j'utilise platformIO mais Arduino IDE fonctionne aussi en théorie

Sommaire

1. Bases du C++
2. Spécificités de la prog embarquée
3. Plus de librairies si jamais vous en avez besoin

Bases du C++

Tout programme C est un programme C++

Donc :

- `#include <stdint.h>` fonctionne sans problème
- vous pouvez coder comme en C

Mais je massacre personnellement la première personne que je vois utiliser `malloc`.

Références

Quand on passe une variable par référence à une fonction, **on lui donne la variable exacte**, et non une copie.

Cela permet de **ne pas utiliser de pointeurs pour modifier une variable**.

En général, on préfère utiliser `f(const int &x)` que `f(int x)` pour éviter de créer des variables locales inutiles.

```
void doubler_ptr(int *ptr_n) {
    *ptr_n = (*ptr_n) + (*ptr_n);
}
void doubler_ref(int &n) {
    n = n + n;
}
```

Utilisation :

```
int n = 4;
doubler_ptr(&n);
doubler_ref(n);
std::cout << n << "\n"; // 16
```

Tableaux

Remarque : quand on est sur un PC il est conseillé d'utiliser les librairies `<vector>` et `<array>` de la STL au lieu d'un tableau.

Les tableaux fonctionnent comme en C.

```
int arr1[2] = {1, 2};
std::array<int, 10> arr2 = {3, 4};
```

Allocation dynamique

Rappel : l'allocation est dynamique si le choix de la quantité de mémoire allouée se fait à l'exécution du programme.

```
int n = rand() % 100;

int *arr_c = (int *) malloc(n * sizeof(int)); // C

int *arr_cpp = new int[n]; // C++
```

On a : `type* variable = new type[taille]` pour un type quelconque. `[taille]` est optionnel.

Destruction dynamique

Pour détruire une variable créée avec `new` on utilise `delete`.

```cpp
int *ptr_cpp = new int;
int *arr_cpp = new int[n];

delete ptr_cpp; // suppression de pointeur
delete [] arr_cpp; // suppression de tableau
```

Programmation embarquée

Fondamentalement une philosophie différente par rapport à nu programme classique.

- on utilise un minimum de librairies
- pas de tableaux dynamiques si on peut éviter
- les variables globales c'est bien
- **ne pas oublier l'architecture physique**

Programme n°1 - au boulot

Créer un programme qui envoie du texte sur votre terminal (Arduino).

```
Serial.begin(uint32_t baud_rate);
Serial.print(...); // variantes : println, printf
delay(uint32_t time_ms);
```

Note : baud_rate par défaut 9600, pour le changer ajouter `monitor_speed = 115200` dans *platformio.ini*

Bonus : si vous cous faites chier, lire un mot tapé au clavier puis print ce qui a été tapé

Programme n°1 - points intéressants

- Quel protocole de communication a été utilisé ?
- c koi le baud rate ?
- Ca risque de bloquer quels pins d'utiliser cette méthode d'envoyer des données ?

Pins - généralités

Chaque pin de la STM32 peut avoir plusieurs fonctions, mais une seule à la fois.

On les retrouve dans le **pinout** (trouvé en une recherche google).

Il est important de **toujours savoir quels pins sont utilisés par un programme**. Par exemple en les définissant tous dans `pins.h`

Pins - modes de fonctionnement

- fonction spécifique (ex: SDA, TX, PWM, ADC...)
- output (0 ou 3.3V sur une STM32, 5V sur certaines cartes)
- input flottant (on évite)
- input pullup
- input pulldown

Programme n°2 - au boulot

Prendre un bouton. Faire en sorte d'afficher son état sur le terminal.

```
pinMode(uint32_t pin, uint32_t mode); // mode : INPUT, OUTPUT, INPUT_PULLUP...
digitalRead(uit32_t pin);
```

Bonus : brancher une LED (**ne pas oublier la résistance**), le bouton change l'état de la LED a chaque fois qu'on appuie dessus.

La STL

La STL est l'une des plus grandes différences entre le C et le C++.

Un grand nombre d'algorithmes et structures de données y figurent.

Les plus utiles sont :

```
#include <array> // Tableaux
#include <string> // Chaînes de caractères
#include <vector> // Tableaux de taille variable
#include <map> // Dictionnaires python
#include <list> // listes chaînées
#include <algorithm> // mais juste la fonction sort()
```

<array>

```cpp
array<int, 5> arr = {1, 2, 7, 4, 3};
cout << "arr[2] = " << arr.at(2) << "\n"; // 7
cout << "Size = " << arr.size() << "\n"; // 5
cout << "First element = " << arr.front() << "\n"; // 1
cout << "Last element = " << arr.back() << "\n"; // 3

sort(arr.begin(), arr.end()); // on trie

cout << "Sorted arr : ";
for(int val : arr) { // la magie du C++, boucle sur les éléments
    cout << val << " ";
}
cout << endl; // Sorted arr : 1 2 3 4 7
```

Le reste

RTFM

https://en.cppreference.com/w/

Bonne chance, la semaine prochaine **interruptions et timers**.