S4 – Développement logiciel

Programmation orientée objet

Programmation orientée objet avec Arduino
Créer une classe pour Arduino

Objectifs : être capable de créer et d'utiliser une classe pour Arduino .

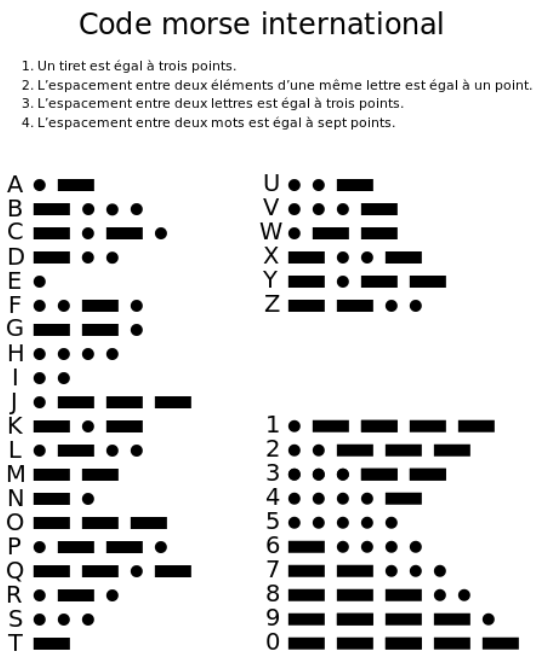
Introduction

L'objectif de ce TP est de mettre en pratique les notions abordées lors de l'étude de la programmation orientée objet. Vous allez créer une classe pour Arduino et l'utiliser. Vous écrirez tous les fichiers nécessaires à l'utilisation de cette classe : fichier en-tête (fichier .h), fichier de définition de classe (fichier .cpp), fichier d'exemple et un fichier mot-clé permettant la coloration syntaxique des éléments créés. Tous ces fichiers seront regroupés dans un répertoire « prêt à l'emploi ».

Tutoriel (TuTuTu ToToTo TuTuTu)

1. *Le code morse*

Le code morse ou alphabet morse, est un code permettant de transmettre un texte à l'aide de séries d'impulsions courtes et longues, qu'elles soient produites par des signes, une lumière ou un geste. En code morse, S.O.S s'écrit : … --- … (C'est bon à savoir, on ne sait jamais, ça peut toujours servir !).



Code morse (image wikipedia)

La plus grande partie de ce tutoriel est traduite et adaptée (librement) d'un document en anglais disponible sur le site Arduino : « Writing a library for Arduino » à l'adresse suivante : https://www.arduino.cc/en/Hacking/LibraryTutorial

2. Implémentation « normale »

Voici une implémentation « normale » (sous entendu sans utiliser de classe mais en utilisant des fonctions) d'un programme émettant avec une led un SOS en code morse.

```
int pin = 13;

void setup()
{
  pinMode(pin, OUTPUT);
}

void loop()
{
  point(); point(); point(); // S
  delay(500); // espace lettre
  trait(); trait(); trait(); // O
  delay(500); // espace lettre
  point(); point(); point(); // S
  delay(1500); // espace "mot"
}

void point()// point ou court .
{
  digitalWrite(pin, HIGH);
  delay(250);
  digitalWrite(pin, LOW);
  delay(250);
}

void trait()// trait ou long -
{
  digitalWrite(pin, HIGH);
  delay(750);
  digitalWrite(pin, LOW);
  delay(250);
}
```

Travaux dirigés 1

Q.1) Représenter sur un chronogramme l'état de la led au cours du temps sur un cycle.

Q.2) Vérifier la conformité des signaux avec le code morse tel que décrit sur la figure « code morse ».

Q.3) Détaillez le rôle des fonctions utilisées.

3. *Écriture de la classe*

Nous allons maintenant implémenter certaines parties de ce programme dans une classe nommée Morse.

Nom de la classe	**Morse**	
Attributs	- _pin	
méthodes()	+ Morse(int pin) + point() : void + trait() : void	Constructeur de la classe

« + » signifie public ; « - » signifie privé

L'attribut (privé) de cette classe est _pin qui représente la broche où sera connectée la led.
Les méthodes (publiques) sont Morse(int pin) le constructeur de la classe (voir ci dessous), point() la méthode qui permet d'allumer la led pendant la durée d'un point et trait(), la méthode qui permet d'allumer la led pendant la durée d'un trait (ou tiret).

Il s'agit de créer maintenant a l'aide de Notepad++ (vous pouvez aussi utiliser Code::Blocks) le fichier en-tête (ou header) appelé Morse.h (avec une majuscule).

```
#ifndef Morse_h
#define Morse_h
#include "Arduino.h"

class Morse
{
  public:
    Morse(int pin);
    void point();
    void trait();
  private:
    int _pin;
};

#endif
```

On rappelle : les deux premières lignes et la dernière ligne permettent d'éviter des boucles d'inclusion infinies.

La ligne #include "Arduino.h" permet d'avoir accès aux variables et constantes spécifiques à Arduino.

Une classe doit posséder une méthode spécifique (une fonction) appelée constructeur qui est chargé d'initialiser une instance (objet) de la classe. Le constructeur est appelé systématiquement au moment de la création de l'objet, il porte le même nom que la classe et ne retourne aucun type.
Les méthodes point() et trait() sont définies en public.

L'attribut _pin est déclaré comme privé, cette variable contiendra le numéro du pin où la led est

câblée. On utilise généralement le caractère _ pour distinguer l'appellation de l'attribut de l'argument de la fonction.

Il faut maintenant créer un nouveau fichier appellé « Morse.cpp ».
Ce fichier contiendra la description des méthodes.

```
#include "Arduino.h"
#include "Morse.h"

Morse::Morse(int pin)
{
  pinMode(pin, OUTPUT);
  _pin = pin;
}

void Morse::point()
{
  digitalWrite(_pin, HIGH);
  delay(250);
  digitalWrite(_pin, LOW);
  delay(250);
}

void Morse::trait()
{
  digitalWrite(_pin, HIGH);
  delay(750);
  digitalWrite(_pin, LOW);
  delay(250);
}
```

Le fichier contient sur les deux premières lignes : #include "Arduino.h" (accès au fonctions Arduino) et #include "Morse.h" (définition de l'en-tête associé).
Puis vient la définition des méthodes.
Toutes les méthodes commencent par Morse ::
On retrouve le constructeur (qui effectue aussi l'initialisation de la broche en sortie) puis les deux méthodes (point et trait) qui correspondaient aux fonctions dans le cas d'une « implémentation normale ».
Remarquez les tours de passe-passe entre pin et _pin. Les méthodes ne « manipulent pas » pin mais sa « réplique » privée _pin ! _pin prend la valeur de pin au moment de l'instanciation de l'objet.

Manipulation n°1

Q.1) Tester et validez le premier programme sur Arduino (implémentation « normale » c'est à dire sans utiliser de classes.

Manipulation n°2

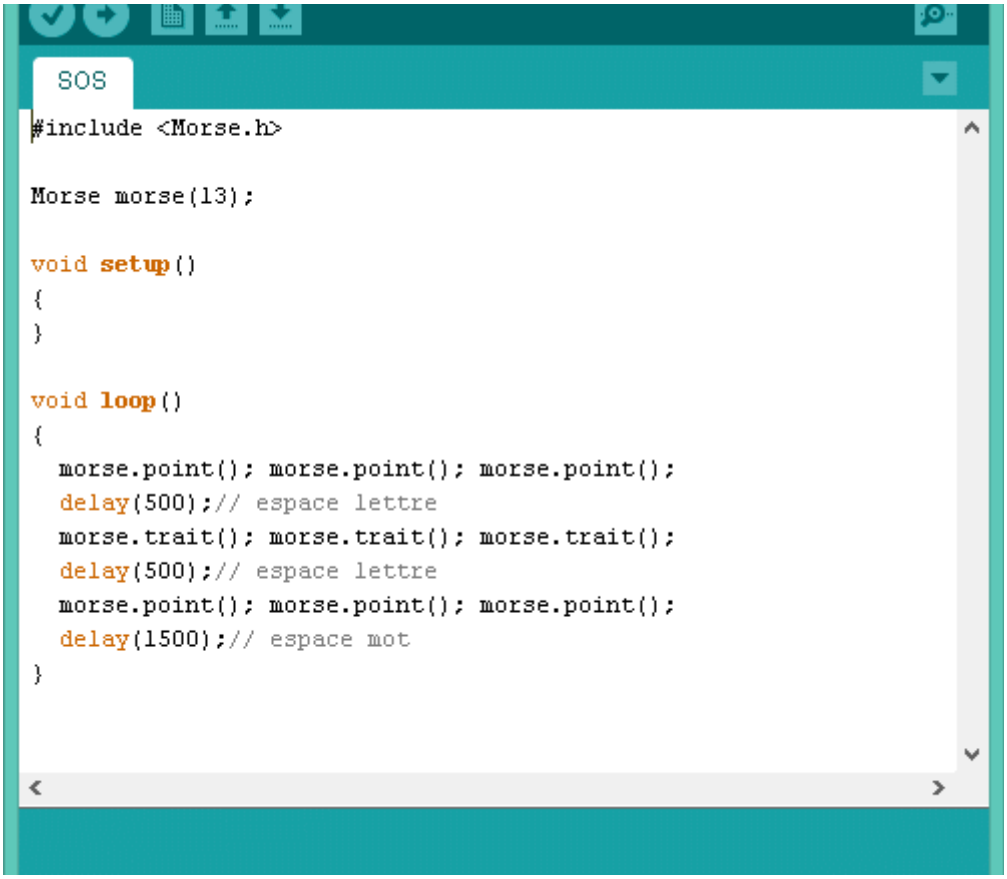
En suivant les instructions du tutoriel, vous allez :

Q.1) Créer les fichiers Morse.h et Morse.cpp .

Q.2) Placer vos deux fichiers dans un répertoire Morse puis déplacer ce répertoire dans le repertoire « libraries » d'Arduino.

Q.3) Relancer Arduino et vérifier que le répertoire est bien vu par Arduino en effectuant l'opération Croquis > Importer bibliothèque.

Q.4) Saisir le programme SOS permettant de tester la classe réalisée. Celui ci est donné :


SOS

```
#include <Morse.h>

Morse morse(13);

void setup()
{
}

void loop()
{
  morse.point(); morse.point(); morse.point();
  delay(500);// espace lettre
  morse.trait(); morse.trait(); morse.trait();
  delay(500);// espace lettre
  morse.point(); morse.point(); morse.point();
  delay(1500);// espace mot
}
```

Q.5) Vérifier le bon fonctionnement du programme de test !

4. Les « finitions »

Le programme est opérationnel, on peut encore améliorer la convivialité de cette classe.

Fichier exemple

Il est agréable lorsque l'on récupère une classe (une bibliothèque) d'avoir des exemples de mise en œuvre de cette classe.

Il suffit de placer le répertoire SOS contenant le fichier SOS.ino (programme Arduino) dans un répertoire « examples » dans votre répertoire Morse.
Je récapitule :
Arduino>Libraries>Morse>examples>SOS>SOS.ino
Morse.h et Morse.cpp etant placés dans Arduino>Libraries>Morse>
Relancer Arduino après l'opération.

Manipulation n°3

Q.1) Vérifier que le fichier SOS d'exemple est bien accessible à partir du menu Arduino Fichier>Exemples>Morse

Coloration syntaxique

Pour finir, notre classe manque un peu de couleur. Il est possible d'obtenir la coloration syntaxique du programme réalisé en réalisant un fichier nommé keywords.txt.

```
1   Morse    KEYWORD1
2   trait    KEYWORD2
3   point    KEYWORD2
```

Chaque ligne porte le nom d'un mot clé suivi d'une tabulation (pas d'espaces) suivi de KEYWORD1 pour une classe ou de KEYWORD2 pour une méthode. La classe sera ainsi coloré en orange, les méthodes en marron (la différence de couleur est faible). Il faut là aussi re-démarrer Arduino pour que la modification soit effective.

Manipulation n°4

Q.1) Vérifier que tout fonctionne bien avec la coloration syntaxique : noms de la classe et des méthodes en couleur !

Ma led a la classe !

Maintenant, à vous de jouer. Nous allons commencer par réaliser une classe très simple : une classe permettant d'allumer une led, de l'éteindre ou de la faire clignoter.

Voici la définition de la classe :

Nom de la classe	**Led**	
Attributs	- _pin - _etat	
méthodes()	+ Led(int pin) + ledOn() : void + ledOff() : void + ledClign(int T, int N) : void + ledChange() : void + getEtat() : bool	Constructeur de la classe

Définition des attributs

_ pin correspond au numéro de la broche
_etat correspond à l'état de la led (false : led éteinte ; true : led allumée)

Définition des méthodes

Led(int pin) : le constructeur de la classe
ledOn() : allume la led et renseigne son état
ledOff() : éteint la led et renseigne son état
ledClign(int T, int N) fait clignoter la led N fois, la led est allumée de 0 à T/2, éteinte de T/2 à T.
ledChange() allume la led si elle est éteinte, éteint la led si elle est allumée et renseigne l'état de la led
getEtat() permet de récuperer l'état de la led

Remarque : ledClign(int T,int N) et ledChange() pourront utiliser ledOn() et LedOff().

Manipulation n°5

Q.1) Écrire tous les fichiers (Led.h, Led.cpp, fichier exemple, fichier keyword) permettant de faire fonctionner cette classe et de valider son bon fonctionnement. Votre fichier exemple devra tester toutes les méthodes et utilisera le moniteur série (en particulier) pour valider la méthode getEtat().

Bon courage !!

Arduino la Classe !

Non, lui c'est pas Arduino la classe, c'est Aldo la classe, c'était un personnage à la mode dans les années 80. Vous êtes trop jeunes, vous pouvez pas comprendre, mais c'est drôle !

Retrouvez d'autres cours et documents sur :
http://www.louisreynier.com