

Mise en situation et objectifs du TP

On abordera l'utilisation des fonctions et le passage de paramètres.

Avec ce TP on verra comment organiser un projet Arduino. On utilisera des fichiers d'entête et la répartition du code source d'un projet sur plusieurs fichiers.

Consignes

- Vous avez créé un dossier dans votre espace de travail pour le TP1 et TP2 dans **TP_Arduino** c'est là que vous devez continuer à enregistrer vos projets Arduino pour ce TP dans un dossier TP3.
- Tous les fichiers (*.ino *.h) des différents exercices devront être sauvegardés dans un dossier ayant le même nom que le fichier principal du projet Arduino, c'est une obligation avec l'IDE Arduino, cette opération est effectuée lors de la sauvegarde.
- Si des fichiers (*.ino *.h) sont utilisés dans plusieurs projets il seront systématiquement recopiés dans les dossiers des projets.

Seuls les fichiers sauvegardés permettront de noter votre travail.

1) Utilisation des fonctions	2
1.1) Notion de fonction	2
1.2) paramètres d'une fonction	2
1.3) Ecriture d'une fonction	3
1.4) Applications.....	3
2) Organisation d'un projet Arduino.....	5
2.1) Conventions typographiques	5
2.1.1) Indentation.....	5
2.1.2) Commentaires	5
2.1.3) Les noms de fonction	5
2.1.4) Les constantes	5
2.1.5) Les variables	5
2.2) utilisation de plusieurs fichiers sources	6
2.3) Ajout d'un fichier INO ou H au projet Arduino	6
2.4) Applications.....	6
2.4.1) Projets multi fichiers	6

90% of coding is debugging.
The other 10% is writing bugs.

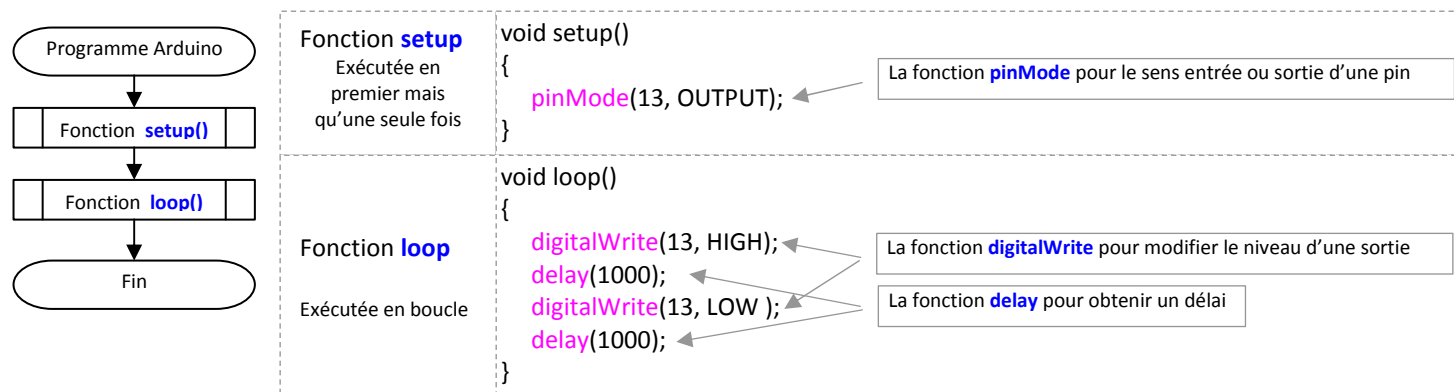
1) Utilisation des fonctions

1.1) Notion de fonction

On appelle fonction un sous-programme qui permet d'effectuer un ensemble d'instructions. Une fonction peut être appelée par le programme principal ou par une autre fonction. Les fonctions qui s'appellent elles mêmes (fonction récursive) sont plus délicates à manipuler et ne seront pas abordées dans ce TP.

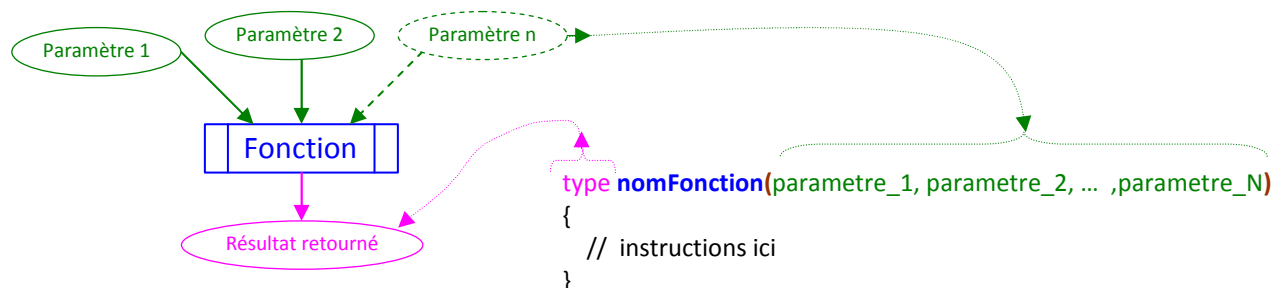
L'utilisation des fonctions permet de ne réécrire les mêmes instructions plusieurs fois dans un programme cela simplifie le code et réduit la taille des programmes.

Il existe des fonctions déjà disponibles dans les bibliothèques Arduino, il est aussi souhaitable de créer des fonctions lors de l'écriture d'un projet, cela permet de rendre les programmes plus lisibles et facilite la mise au point.



1.2) paramètres d'une fonction

Une fonction permet d'effectuer un ensemble d'instructions pour cela on peut lors de son appel lui passer des paramètres avec lesquels elle va travailler et en retour elle pourra renvoyer le résultat de son travail.



- **type** : correspond à la sortie, c'est le type de la fonction. Comme les variables, les fonctions ont un type. Ce type dépend du résultat que la fonction renvoie : si la fonction renvoie un caractère, vous mettrez `char`, si elle renvoie un entier vous mettrez `int` ou `long`, etc. Mais il est aussi possible de créer des fonctions qui ne renvoient rien, dans ce cas le type est « `void` ».
- **nomFonction** : c'est le nom de votre fonction. Vous pouvez appeler votre fonction comme vous voulez, du temps que vous respectez les mêmes règles que pour les variables (pas d'accents, pas d'espaces, etc.). le nom ne doit pas comporter d'espaces, vous pouvez utiliser pour cela le signe « `_` ».
- **paramètres** : correspond à l'entrée, entre parenthèses vous pouvez envoyer des paramètres à la fonction. Ce sont des valeurs avec lesquelles la fonction va travailler, ces paramètres doivent être séparés par une virgule. Il faudra indiquer le type du paramètre (`char`, `int`, `long`, `float`,... etc.). Vous pouvez envoyer autant de paramètres que vous le voulez. Vous pouvez aussi n'envoyer aucun paramètre à la fonction, dans ce cas on écrira « `void` » entre les parenthèses.

Exemple pour une fonction permettant d'additionner deux nombres et renvoyer le résultat →

Pour que la fonction puisse renvoyer son résultat on utilise le mot réservé « **return** » suivi par la valeur ou variable à renvoyer. Le mot « **return** » permet aussi de quitter la fonction, il sera en général situé en fin de fonction.

```
int addition( int a, int b )
{
  return a + b;
}
```

1.3) Ecriture d'une fonction

Construction de la fonction	Arguments passés	Valeur en retour	Utilisation
void fonction1 (void) { s = a + b ; }	aucun	aucune	fonction1() ; s, a et b sont des variables globales
void fonction2 (int a , char b) { s = a + b ; }	1 int 1 char	aucune	fonction2(var1, var2) ; a et b sont des variables locales s est une variable globale
int fonction3 (void) { return a + b ; }	aucun	int	var3 = fonction3() ; a et b sont des variables globales
int fonction4 (int a , char b) { return a + b ; }	1 int 1 char	int	var4 = fonction4(var1, var2) ; a et b sont des variables locales

Il est souhaitable de déclarer le prototype de la fonction avant de l'utiliser c'est-à-dire l'appeler.

Cette déclaration n'est pas obligatoire avec l'Arduino.

Prototype d'une fonction	Fonction	Appel d'une fonction
int somme (int a , char b) ;	Prototype à écrire avant utilisation de la fonction. Il faut mettre un point virgule en fin de ligne, les paramètres sont séparés par une virgule.	
int somme (int a , char b) { return a + b ; }	Construction de la fonction. Pas de point virgule en fin de ligne.	
s = somme (nombre1 , nombre2) ;	Appel de la fonction. Si nombre1 = 3 nombre2 = 8 alors s = 11 Il faut mettre un point virgule en fin de ligne les paramètres sont séparés par une virgule.	

1.4) Applications

- Q1 Analysez le programme ci-dessous et expliquez les lignes signalées par « // » (lignes 4,5, 7, 11, etc.)

☐ Validation

- Q2 Saisir, compiler et tester ce programme « **jeudes1** ».

Jeudes1.ino

☐ Validation

- Q3 Quel est le rôle des lignes 32 et 33.

① Pour vérifier leur action ne pas les effacer mais simplement les mettre en commentaire.

☐ Validation

- Q4 Modifier le programme « **jeudes1** » en « **jeudes2** » (enregistrer sous.. « **jeudes2**») avec 4 fonctions pour :

- Initialiser les pins et de le générateur aléatoire.
- Initialiser la liaison avec la console et afficher le titre du jeu.
- Afficher la valeur du dès.
- Faire clignoter la led en fonction de la valeur du dès.

Jeudes2.ino

☐ Validation

- Q5 On désire obtenir l'affichage ci-dessous. Créer une fonction pour ce nouvel affichage avec un programme « **jeudes3** » (« **jeudes2**» enregistrer sous.. « **jeudes3**»).

Jeudes3.ino

☐ Validation

Au départ	1	2	3	4	5	6
jeu de DES V3	Valeur : 1	Valeur : 2	Valeur : 3	Valeur : 4	Valeur : 5	Valeur : 6
Valeur : 0	Valeur : 1	Valeur : 2	Valeur : 3	Valeur : 4	Valeur : 5	Valeur : 6
---	---	* --	* --	* _ *	* _ *	* _ *
---	- * -	---	- * -	---	- * -	- * -
---	---	-- *	-- *	* _ *	* _ *	* _ *

Les caractères « - » et « * » seront séparés par un espace.

1	/*	
2	Jeu de des 1	
3	*/	
4	#define LED 9	//
5	#define BP 4	//
6		
7	long des;	//
8		
9	void setup()	
10	{	
11	pinMode(LED, OUTPUT);	//
12	pinMode(BP, INPUT);	//
13	Serial.begin(38400);	//
14	Serial.println("jeu de DES V1");	//
15	randomSeed(analogRead(0));	//
16	}	
17		
18	void loop()	
19	{	
20	if(digitalRead(BP) == 0)	//
21	{	
22	des = random(1, 7);	//
23	do delay(100);	//
24	while (digitalRead(BP) == 0);	//
25	Serial.print("Valeur : ");	//
26	Serial.println(des, DEC);	//
27	while(des)	//
28	{ digitalWrite(LED, HIGH);	//
29	delay(200);	//
30	digitalWrite(LED, LOW);	//
31	delay(300);	//
32	des--;	//
33	}	
34	}	
35	}	

2) Organisation d'un projet Arduino

2.1) Conventions typographiques

2.1.1) Indentation

L'indentation consiste à ajouter des tabulations ou des espaces en début de ligne. On doit indenter son code pour les raisons suivantes :

- Il améliore grandement la lisibilité du code
- Il permet de détecter des erreurs de syntaxe
- Il rend le code plus esthétique

En langage C, et donc avec l'Arduino, l'indentation n'est pas obligatoire contrairement au Python par exemple. Toutefois **l'indentation est très vivement conseillée**.

Le style d'indentation décrit les différentes manières que les programmeurs utilisent pour faire ressortir un bloc de code. Il existe plusieurs styles d'indentation d'un code source C, un exemple est donné ci contre.

```
void a_function(void)
{
    if (x == y)
    {
        fonction1();
        fonction2();
        if (x > z)
        {
            fonction3();
            fonction4();
        }
    }
    fonction_x();
}
```

Diagram illustrating code blocks and function scope:

- Fonction**: The entire function scope, indicated by a long vertical double-headed arrow.
- Bloc 1**: The first nested block (the `if (x == y)` block), indicated by a medium vertical double-headed arrow.
- Bloc 2**: The second nested block (the `if (x > z)` block), indicated by a short vertical double-headed arrow.

2.1.2) Commentaires

Il est souhaitable de commenter mais de ne commenter que ce qui apporte un supplément d'information.

Le choix judicieux du nom des identificateurs, variables, constantes, nom des tableaux, nom des fonctions facilite la compréhension du code écrit.

Pour un commentaire long utiliser « `/* ... */` » sur plusieurs lignes. Les commentaires en fin de ligne risquent lors de l'impression de rendre l'ensemble plus difficilement lisible, ce qui n'est pas le but recherché, il est alors possible de placer le commentaire au dessus du code.

2.1.3) Les noms de fonction

Il est d'usage d'utiliser les minuscules. Il existe des cas où l'on a besoin de 2 mots pour nommer une fonction. Il n'est évidemment pas question d'accoler ces deux mots en minuscule, car le code serait vite illisible. Il existe plusieurs possibilités :

1. Coller les mots en mettant une majuscule au début de chaque mot : **CommandeLed()**
2. Coller les mots en mettant une majuscule que pour le 2ème mot : **commandeLed()**
3. Séparer les mots en mettant un souligné « `_` » (underscore) entre chaque mot : **commande_led()**

Il est souhaitable que le nom de la fonction donne une indication claire du rôle de cette fonction cela évitera l'ajout d'un commentaire. Une fonction peut utiliser les caractères suivants : « A...Z, a...z, 1...9, `_` », toutefois une fonction ne doit jamais commencer par un chiffre ni utiliser un mot réservé du langage (main, void, for, .. Etc.)

2.1.4) Les constantes


Il est d'usage d'utiliser les MAJUSCULES, le C étant sensible à la case ne pas mélanger majuscules et minuscule simplifiera l'écriture. Pour le nom mêmes restriction que pour les variables.

#define	PI	3.1416	Utilisation de la macro #define. Pas d'espaces, utiliser l'underscore pour faire un espace.						
#define	LED_BUILTIN	13							
#define	POUSSOIR	4							
const	float	PI	=	3.1416;	Utilisation d'une "variable non modifiable", les valeurs sont typées (ici un float). A0 est une constante déjà définie dans Arduino				
const	int	CAPTEUR	=	A0;					
Quelques constantes définies dans Arduino			HIGH	INPUT	true	LOW	OUTPUT	false	Attention respecter la casse.

2.1.5) Les variables

Le nom de la variable doit donner une indication claire sur le rôle de cette variable. Une variable peut utiliser les caractères suivants : « A...Z, a...z, 1...9, `_` », toutefois une variables ne doit jamais commencer par un chiffre ni utiliser un mot réservé du langage (main, void, for, .. Etc.)

Il est d'usage d'utiliser les minuscules, le C étant sensible à la case ne pas mélanger majuscules et minuscule simplifiera l'écriture. Il est déconseillé d'écrire 2 variables différentes avec le même nom mais l'une en majuscule l'autre en minuscule. L'utilisation par erreur de l'une à la place de l'autre ne sera pas détectée comme une erreur, la case étant respectée.

Possible	Déconseillé
unsigned int somme ; unsigned int moyenne_mesures ; unsigned int moyenneMesures ;	unsigned int somme ; unsigned int SOMME ; 

2.2) utilisation de plusieurs fichiers sources

Pour les programmes simples les fichiers Arduino sont ainsi organisés : un seul fichier (*.ino) contient l'ensemble du code, il est contenu dans un dossier du même nom.

Lorsque le projet est important il est commode de répartir le code dans plusieurs fichiers.

Si le projet comporte plusieurs fichiers source (*.ino) chaque fichier comporte son fichier « header » (*.h) ou fichier d'entête.

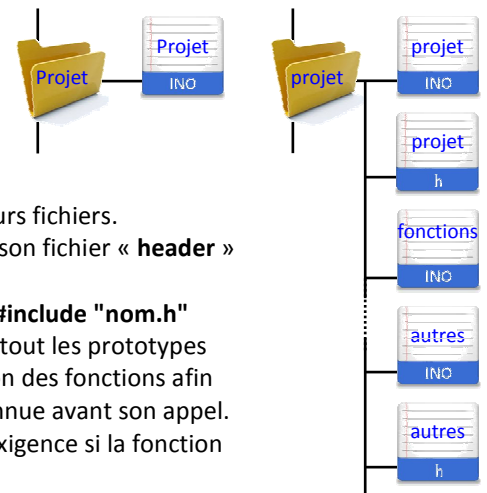
Ce fichier sera inséré au début du fichier INO par la directive de compilation **#include "nom.h"**

Ce fichier pourra comporter les définitions des pins d'entrée et de sortie et surtout les prototypes des fonctions. Ces prototypes permettent aux autres fichiers de connaître la constitution des fonctions afin qu'elles puissent être utilisées, en effet en C la constitution d'une fonction doit être connue avant son appel.

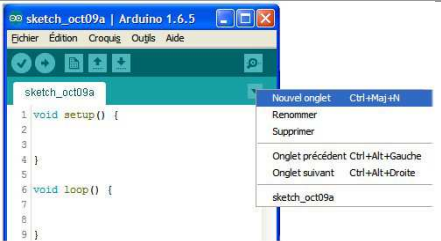

Le fait de placer son prototype en début de fichier permet de satisfaire cette exigence si la fonction est dans un autre fichier. **Avec Arduino cette précaution n'est pas nécessaire.**

Il peut arriver que le lien avec les symboles des bibliothèques Arduino ne se fasse pas correctement.

Dans ce cas là, rajoutez l'include suivant au début de votre .h ou .ino : **#include "Arduino.h"**.



2.3) Ajout d'un fichier INO ou H au projet Arduino

 <p>1</p> <ul style="list-style-type: none">▪ Cliquer sur l'icône▪ puis choisir « Nouvel onglet »	<p>2</p> <p>Nom du nouveau fichier : </p> <p>Entrer le nom du fichier *.ino ou *.h</p> <p>Précisez l'extension « .h » ou « .ino »</p> <p>3</p> <p>Nom du nouveau fichier : fonctions.ino</p> <p>Appuyer sur OK pour créer le fichier</p>	 <p>4</p> <p>Le nouvel onglet est ouvert</p> <ul style="list-style-type: none">▪ L'extension INO n'est pas précisée▪ L'extension est précisée pour les fichiers H
--	---	---

2.4) Applications

2.4.1) Projets multi fichiers

- Q1 Reprendre le projet jeudes3, et après avoir validé son fonctionnement, créer le projet « **jeudes4** » (« **jeudes3** » enregistrer sous.. « **jeudes4** »). Créer un fichier « **fonctions.ino** » dans lequel on placera toutes les fonctions sauf setup() et loop() qui resteront dans « jeudes4.ino ».

☐ Validation

Jeudes4.ino