http://www.didel.com/ info@didel.com

www.didel.com/diduino/Cours04.pdf

Cours Arduino/C 4ème partie - Fonctions

Voir www.didel.com/diduino/Liens.pdf pour la table des matières et l'état d'avancement

Les fonctions sont essentielles à une programmation lisible, et mises dans des librairies, elles permettent des programmes courts centrés sur l'application.

4.1 Fonction simple

L'instruction `delay (duree);` est une fonction avec un paramètre. Ce paramètre est une variable 16 bits de type unsigned int ou un nombre décimal entre 0 et 65535.

Prenons un exemple. Si on joue avec des sons, on doit souvent écrire les 4 instructions suivantes, qui ont comme paramètre la demi-période `dper` (500 us pour 1 kHz)

```
int dper = 500;
HP = HPOn ;
delayMicroseconds (dper) ;
HP = HPOff ;
delayMicroseconds (dper) ;
```

On voudrait écrire comme pour un délai `pulseHP (1000) ;` avec la période, et non pas la demi-période en paramètre.

La fonction qui permettra cette écriture s'écrit

```
void pulseHP (u8 p)
{
      HP = HPOn ;
      delayMicroseconds (p/2) ; // div par 2
      HP = HPOff ;
      delayMicroseconds (p/2) ;
}
```

Pourquoi void ? Il n'y a pas de paramètre en sortie, ce champ est vide.

Pourquoi (u8 p) avec le nom de la fonction ?

Là, il faut bien comprendre que p est un paramètre local, un tiroir vide, et que l'on ne retrouvera pas p dans le programme principal, où il ne faut pas le déclarer. Quand on appellera pulseHP on passera un paramètre global, constant ou variable, qui ira dans cette case réservée.

Pourquoi `p/2` ? Le paramètre dans pulseHP est tout naturellement la période, que l'on obtient par deux retards d'une demi-période. Donc il faut diviser le paramètre par 2, ce qui est une opération simple en binaire (diviser pas 3 prendrait beaucoup plus de temps).

Vous préférez que la fonction utilise comme paramètre une fréquence f en Hz? Le paramètre p est remplacé par 1000000/f Donc 1 kHz donne p= 1000000/1000 =1000. Mais quelle sera la durée de la division à faire chaque fois? Si c'est des dizaines de microsecondes, cela s'entendra!

Pour avoir un bip on répète cette impulsion dans une boucle for

```
for ( int i=0; i<400; i++)
{
   pulseHP (1000); // sera exécuté 400 fois, donc 0.4 secondes
}
```

3.6 Exemple sans paramètre

Ce bip que l'on vient de voir peut devenir une fonction. Il suffit d'encadrer les instructions par

```
void bip ()
   {
        for ...
   }
```

et dans le programme, toutes les fois que l'on veut un bip, on écrit `bip();`

3.7 Exemples avec deux paramètres en entrée

Une note simple est une répétition d'impulsions sur le haut-parleur. Le nombre d'impulsions sera le 2e paramètre de la procédure `note (periode, nbDeRepet) ;`

Le produit des deux paramètres donne la durée, mais notre propos ici n'est pas de faire de la musique, mais expliquer les fonctions. La fonction `note` a besoin de deux variables locales, la période `p` et la durée `n`.

```
void note (int p, int n )
{
   for (int i=0; i<n; i++)
   {
       pulseHP (p)
   }
}
```

Pour jouer un ré noir par exemple, il faut connaître la période et durée en nombre d'oscillations. On passe par une table et on écrit pour avoir cette note

```
note (taPeriodes [re], taDurees [re]);
```

Fonction avec un paramètre en sortie

Un exemple bien connu et la fonction prédéfinie `analogRead () ;` qui a un paramètre en entrée, le numéro de la pin, et un paramètre en sortie, la valeur lue.
Si on veut une procédure qui lit un potentiomètre, sans devoir chaque fois dire comment il a été câblé (on doit naturellement le dire une fois dans les déclarations au début), on écrit

déclaration		utilisation	

	`int valPot ()`
	`{`
	`int p ;`
	`p = analogRead () ;`
	`return p ;`
	`}`
			`. . .`
			`if ( valPot() > PotMin )`
			`{`
			`. . .`

On voit que le `void` a été remplacé par le type de la variable qui sera rendue; le compilateur doit réserver la place. Ensuite, après l'accolade, il faut dire le nom de cette variable. On peut alors la préparer (la remplir) et finalement le compilateur veut encore qu'on lui dise qu'elle est emballée et expédiée dans un paquet cadeau.
A l'utilisation, il faut se souvenir que l'on a une variable de type `int` avec laquelle on peut calculer, comparer, etc..
On ne peut retourner qu'une variable, mais il peut y avoir plusieurs paramètres en entrée, dont on va donner le type et le nom local dans la parenthèse suivant le nom de la fonction.

Quelques exemples de fonction utiles

`void AllumePhares (void)`
`{`
`PhareGauche = Allume;`
`PhareDroite = Allume;`
`}`
	Cette fonction peut être beaucoup plus complexe, si on allume les phares progressivement.
	Dans le programme, on écrit
	`AllumePhares () ;`

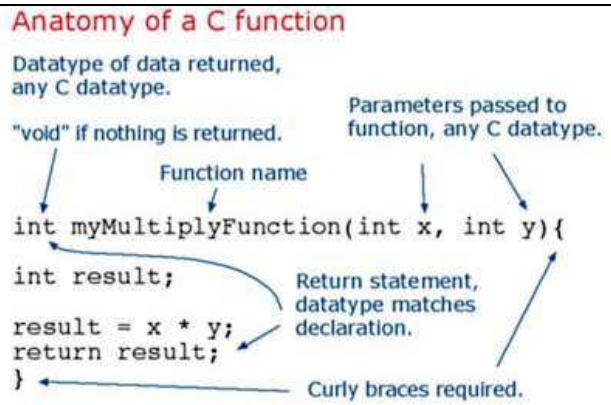
Pour éviter de définir `AllumePhares et EteintPhares`, on pourrait mettre un paramètre qui vaudrait Eteint (0) ou Allume (et avoir l'instruction `SetPhares (Eteint);`

Autres exemples : `void GetPous ()` test avec `KiCarCntPous.pde` dans KiCar.zip
--- a compléter

Pour toute fonction, it faut bien réfléchir aux types de variables, et passer du temps pour choisir les noms, et ne pas hésiter à changer les noms qui deviennent ambigus. Un nom bien choisi évite une tartine d'explications.

Résumé

Ci-contre, ce qu'il suffit de dire pour tout savoir (quand on sait tout) !

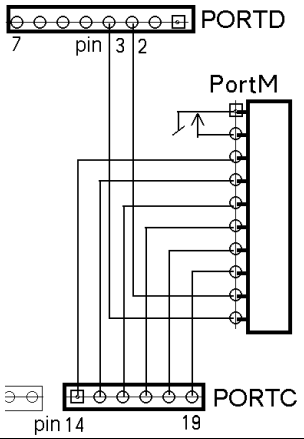


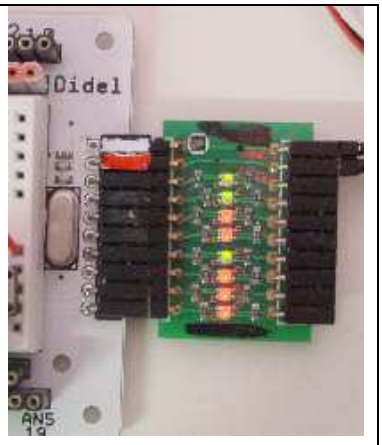
Une fonction est un paquet d'instructions exécutables. Ce paquet porte un nom, ce qui permet de l'appeler, c'est-à-dire provoquer l'exécution des instructions qu'il contient.
On utilise dans les fonctions des variables locales, qui renaissent chaque fois que le bloc est appelé, et meurent à la fin de l'exécution. Pour conserver ces variables, il faut faire précéder leur type du mot `static`, ou les déclarer comme variables globales.

Exemple de fonctions dans www.didel.com/kidules/Librairies.pdf et www.didel.com/kidules/DebugLib.pdf
(adaptation Arduino pas encore faite).

Exemple qui montre aussi comment manipuler des mots binaires

Le connecteur gauche des cartes Diduino est compatible microdules/kidules . Il rassemble deux ports. Pour former une variable dont les bits sont dans l'ordre des pins, il faut lire le PORTC, garder les bits 0 à 5 qui sont à la place qui nous intéresse, puis lire les bits 2 et 3 du PORTD, décaler de 4 positions et superposer, donc sur les bits 6 et 7.
Pour écrire un variable 8 bits (ou 16 bits dont seuls les 8 bits de poids faible sont significatifs), la démarche est inverse.






```
void WritePortM (int b)
{
   PORTC = b;
   PORTD = b<<4;
}
```

Utilisation : `WritePortM (cnt) ;`

```
int ReadPortM (void)
{
  int b ;
  b = PINC & 0x3F ;
  b |= (PIND & 0x0C) << 4 ;
  return b ;
}
```

Utilisation : `ByteLu = ReadPortM() ;`

```
if ( ReadPortM() == 0x24 )
{
   . . .
```

Ces instructions supposent que l'on a compris les opérations logiques. Un Cours0x est prévu. Voir en attendant http://arduino.cc/en/Reference/Boolean