Instructions et commandes C

La série de documents plus récent www.didel.com/coursera/LC1.pdf LC2.pdf etc comporte de nombreux exemples qui complètent bien les notions générales.

Le but de ce document n'est pas d'expliquer les instructions, qui ont été vues dans les exercices, et dont on trouve le détail dans de nombreuses pages sous Google. Le but est d'avoir un résumé sur 4 pages, que le débutant consulte quand il n'est plus sûr de lui (orthographe, syntaxe).
Pour un résumé sur une page,
fonctions www.didel.com/C/Fonctions.pdf
types de données signes et opérateurs www.didel.com/C/Resume.pdf

1 Généralités

1.1 Règles

noms explicites (ne pas utiliser Poussoir – c'est quel poussoir ?)
constantes en majuscule (mots courts) **CR PORTA DODIEZE**
variables en minuscules, majuscule pour séparer (noms explicites) **distanceGauche**
variables locales temporaires, mots courts **i j temp**
fonctions avec majuscule initiale dans un verbe **FaireCeci LireClavier**
exceptions permises si la lisibilité est améliorée.

1.2 Types de données

unsigned char	0..255 0..0xFF
byte (usually accepted)	0..255 0..0xFF
int8_t	0..255 0..0xFF
char uint8_t	-128 .. +127
unsigned int	0..65535 0..0xFFFF
int	-32268 .. 32267
long	0x80000000 .. 0x7FFFFFFF

Une multiplication étend un type 8 bits en 16 bits

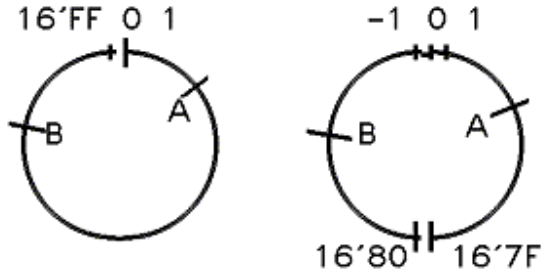


Figure 1

1.3 Numérotation des éléments et poids

Dans une variable ou un tableau, le premier élément est toujours l'élément 0 et pas 1.
Dans un mot binaire, on distingue le rang 7 6 .. 2 1 0 et le poids 128 (=2^7) 64 … 4 2 1

1.4 Déclarer une constante

Le #define permet de donner un nom à une valeur ou une instruction.
Par convention, un nom de constante se note de préférence en majuscule, ce que nous ne faisons pas toujours.

```
Exemples   #define Led1On   bitClear (PORTD,5)
           #define Pous1On  bitClear (PORTD,5)
```

1.5 Include

Include permet au préprocesseur d'insérer un fichier. On utilise des '' '' pour les fichiers locaux, et des < > pour les librairies standard.

```
#include "KiAscDef.h"    même dossier (sketch)
#include <Servo.h>       en librairie servos
```

Au début du 1er include qui utilise des fonctions Arduino, ajouter un #include <Arduino.h>

1.6 Mise en page

Les espaces, tabulateurs et retours à la ligne sont libres, mais il est très important de respecter les règles d'encolonnement avec les tabulateurs.
// devant un commentaire qui se termine à la fin de la lignes
/* devant un bloc de lignes de commentaire qui se terminent par un */

2 Instructions

Les instructions ont un point-virgule terminateur, et on peut mettre plusieurs instructions par ligne (pas recommandé pour la lisibilité et le dépannage). Un `;` seul est vu comme une instruction vide (à éviter)..2.1 Variables

Une variable et son type doivent être déclarés avant utilisation. On peut assigner une valeur initiale par la même occasion. **byte var=12 ;** //Déclare la variable **var** 8 bits positifs et l'initialise à la valeur 12.
On remarque le `;` terminateur des instructions (mais pas des commandes), qu'il ne faut pas oublier. Les espace et lignes supplémentaires sont libres. // ignore la fin de la ligne (commentaire).

Variables globales et locales, static et volatile

Les variables globales sont assignées pour tout le programme. Les variables locales, définies à l'intérieur d'une procédure ne sont pas conservées d'un appel à l'autre.
Pour conserver la variable locale, il faut mettre en premier "static"
Pour éviter que le compilateur optimise en supprimant des accès, il faut utiliser "volatile"

2.2 Assignation (égalité)

On assigne ou change la valeur d'une variable avec une instruction d'égalité.
`var8 = PORTB ; var16 = 3*toto ;` (les multiplications se font en 16 bits au moins)

2.3 Opérations arithmétiques

Les opérations arithmétiques sont le + - * / et %. Les opérations de font en binaire, sur des nombres de 8,16 ou 32 bits, signés ou non. Les dépassement ne sont pas signalés. La division donne un résultat entier, et le reste de la division s'obtient avec l'opération % (modulo).
Les nombres flottants 32 et 64 bits sont supportés par les contrôleurs les plus performants.

2.4 Opérations logiques

Les opérations logiques sont le & (et) | (ou) ^ (ou exclusif) ~ (inversion). Ils agissent sur de mots de 8, 16 ou 32 bits. Pour plus de détails, voir http://www.didel.com/C/OperationsLogiques.pdf

2.5 Comparaison

Si on veut comparer, c'est une opération logique entre des variables qui valent 0 ou différentes de 0, en particulier 1 si la variables est booleenne. On utilise un signe = dédoublé. `if(xx==0)`.
C'est une question que l'on doit mettre entre parenthèse : est-ce que `xx` est égal à `LOW` ? La réponse est vrai ou faux, ''1'' ou ''0''.
`if(xx)` est équivalent à `if(xx==1)` `if (a & b) pour if ((a & b)==1)`
`if(!xx)` est équivalent à `if(xx==0)` `if !(a & b) pour if ((a & b)==0)`
Les opérateurs de comparaison sont **==** (égalité), **!=** (différent), **<** (inférieur), **>** (supérieur), **<=** (inférieur ou égal), **>=** (supérieur ou égal). Ces comparaisons s'appliquent sur des nombres positifs ou négatifs, selon le type de donnée déclaré. Dans la figure 1 plus haut, si A et B sont de type byte (unsigned char), on a A < B , mais si c'est le type char, on a A > B.

Attention, si vous mettez = à la place de ==, le compilateur accepte, mais la condition sera toujours vraie.

2.6 Décalages

Les opérateurs de décalage sont << (décalage à gauche) et >> (décalage à droite).
Pour les nombres positifs des 0 sont injectés
10110101 << 3 donne 10101**000** 10110101 >> 3 donne **000**10110
Pour les nombres signés négatifs des 1 sont injectés dans le décalage à droite (copie du bit de signe)
10110101 << 3 donne 10101**000** 10110101 >> 3 donne **111**10110
On utilise souvent la notation 1<<n pour créer un mot binaire qui a un 1 en n-ieme place
1<<5 → 00100000 1<<0 → 00000001

3 Commandes

Une commande est suivie d'un bloc d'instructions entre `{ }`. S'il y a une seule instruction dans le bloc, les accolades ne sont pas nécessaires, mais c'est source d'erreur car on ajout facilement une instruction en oubliant de remettre les accolades..

3.1 if (condition) { instructions; }

Si la condition est vraie, on exécute le groupe d'instructions qui suit entre accolades.

La condition est en général une comparaison, qui donne comme résultat faux (=0) ou vrai (différent de 0, en général 1).

Exemple	`i++;` // compteur circulaire pour un Dé 0 1 2 3 4 5 0 1 2 … `if(i==6) i=0;`

3.2 if (condition) { } **else** { }
Si la condition est fausse, on exécute le groupe d'instructions qui suit le else.

Exemple	`if (Sens ==0) PORTD = 0b00000010 ;` `else PORTD = 0b00000001 ;`

3.3 if .. else if … else if …else
Avec cette structure, on peut tester plusieurs conditions différentes successivement.
La commande **switch-case** en général mieux adapté et plus lisible.

3.4 while (condition) **{ }**
Le groupe d'instruction entre { } est exécuté jusqu'à ce que la condition devienne fausse. Donc **`while (1) {instructions;}`** boucle indéfiniment !

Exemple	`void main () {`
	`... instructions d'initialisation`
	`while (1) {`
	`... boucle des instructions du programme`
	`}`
	`}`

3.5 do while (condition) **{ }**
La boucle **`do while`** boucle aussi jusqu'à ce que la condition ou l'expression entre les parenthèses ( ) devienne fausse, mais le test se fait après les instructions, qui sont donc exécutées au moins une fois.

Exemple	`do { //on a déclaré avant byte a=0 ;`
	`a++ ;`
	`}`
	`while (a<10); // ; à la fin`

3.6 for
Il y a 3 parties dans l'entête d'une boucle for :

```
for (initialization; condition; increment)
 {
 //instruction(s);
 }
```

L'initialisation a lieu en premier et une seule fois. A chaque exécution des instructions de la boucle, la condition est testée; si elle est VRAIE, le bloc d'instructions et l'incrementation sont exécutés. puis la condition est testée de nouveau. Lorsque la condition devient FAUSSE, la boucle stoppe.

Exemple	for (int i=1; i<=10; i++) { instructions ; } // la variable i est locale

3.7 break;
L'instruction `break ;` permet de sortir d'une boucle if, while, for ou case.
Par exemple, pour sortir de la boucle si on presse sur une touche : if (PousOn) break ;

3.8 Tableaux
Un tableau est une suite de cases mémoires de même taille, accessibles à l'aide d'un numéro d'index. à partir du nom donné. Le tableau a un type , une longueur que l'on mets entre crochet et une suite de valeurs entre accolade.

Exemple	Pour le dé électronique
	`byte faces[6]={0b01000000,0b00…. ;`
	Pour lire la combinaison en position 3 (4^{e} élément !)
	`PORTB = faces[3] ;`

3.9 Structures
Une structure regroupe des valeurs de types éventuellement différents et on peut partir dans des bases de données. Ce n'est pas notre objectif.

3.10 Enumération

Le type **enum** associe des noms à une valeur mémorisée dans une variable. enum est un nom réservé suivi entre accolade d'une suite de symboles qui se voient attribuer les valeurs 0, 1, 2.. qui sont rangées dans une variable dont le type est unique, donc pas spécifié.
Ceci est donc une aide à la documentation.
`enum { Do1, Do2, Do3 } var ;` On peut définir ensuite un `switch (var)`

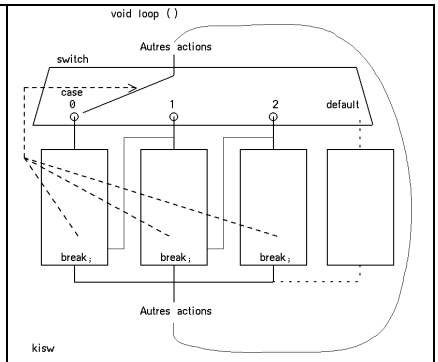
3.11 switch case

La commande `switch` mentionne une variable et est associée aux commandes case qui sélectionnent des valeurs de cette variable.

```
 switch (variable) {
     case val1:
         instruction ;  break ;
     case val1:
         instruction ;  break ;
 . . .
     default:
 . . .
 }
```

Il faut parenthéser le groupe `switch`, mais les `case` et `default` sont terminés par un `:` , sans accolade pour le groupe d'instructions.

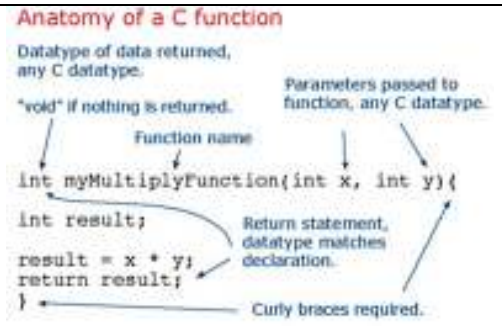
`break ;` sort du groupe `switch`, pour ne pas exécuter les instructions des autres `case`.



4 Fonctions

Une fonction ou procédure peut avoir des paramètres en entrée et au maximum un paramètre en sortie.
Elle agit sur des variables globales, déclarées en dehors de la procédure, et des variables locales déclarées à l'intérieur de la procédure.

Pour une présentation détaillée, voir
www.didel.com/C/Fonctions.pdf



5 Préprocesseur

5.1 3 #define Insertion conditionnelle

nsertion conditionnelle

```
#define Picstar
. . .
#ifdef Picstar
  . . . //instructions valables pour le Picstar
#endif
#ifndef Picstar
  . . . //instructions valables pour les autres cartes
#endif
```

Divers

Vous avez utilisé le type byte, qui n'est pas accepté par certains compilateurs, il ne faut pas changer partout, mais écrire au début #typedef byte unsigned char ou #typedef byte uint8_t (nouvelle norme AINSI)

Noms réservés - ils apparaissent en couleur dans les programmes

auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
inline *(C99)*	int	long	register	restrict *(C99)*	return	short	signed
sizeof	static	struct	switch	typedef	union	unsigned	void
volatile	while						