I Généralités

Le langage C appliqué aux microcontrôleurs permet de bénéficier d'un langage universel et portable pratiquement indépendant du type de microcontrôleur utilisé.

Ceci permet d'éviter :

-les tâches d'écritures en langage assembleur,

-la connaissance de plusieurs langages assembleur dépendant du microcontrôleur utilisé.

II Structure d'un programme en langage C

La structure d'un programme en C est la suivante :

```
// inclusion de fichiers permettant d'utiliser des fonctions (sous programmes) déjà
// créés ou des équivalences correspondant au microcontrôleur utilisé.
// la syntaxe utilisée est le suivante :

#include      <fichier1.h>
#include      <microntrôleur.h>

// déclarations d'équivalences sur des constantes
// la syntaxe utilisée est la suivante :

#define       const1=0x00

// déclarations des variables sous différents formats (8 bits, 16 bits ou 32 bits)
// pour la syntaxe : voir le paragraphe sur la déclaration des variables.

char          val1;
int           val2;
float         val3

// déclarations de toutes les fonctions (sous-programme) utilisées
// le détail des fonctions se trouve après le programme principal

void fonction1(void) ;
void fonction2(void) ;

// programme principal

void main(void)
        {
        …
        …
        }

// détail des fonctions déclarées précédemment

void fonction1(void)
        {
        …
        …
        }

void fonction2(void)
        {
        …
        …
        }
```

III Les variables

Les différents types de variables utilisées en C sont les suivantes :

Syntaxe	Définition	Taille	Plage de variation
char	Variable signée de type caractère ASCII pouvant être utilisée comme un nombre entier signé	8 bits	-128 à 127
unsigned char	Variable non signée de type caractère ASCII pouvant être utilisée comme une nombre entier non signé	8 bits	0 à 255
int	Variable signée de type nombre entier	16 bits	-32768 à 32767
unsigned int	Variable non signée de type nombre entier	16 bits	0 à 65535
float	Variable de type nombre réel	32 bits	$+/- 3,4\times10^{-38}$ à $3,4\times10^{38}$

IV Les opérateurs

Toutes les opérations réalisées se finissent par **;**
Exemple : Z=X+Y ;

IV.1 Les opérateurs arithmétiques

Ces opérateurs permettent d'effectuer les opérations arithmétiques traditionnelle : addition, soustraction, multiplication et division entière entre des variables.

Opérateur	Fonction
+	Addition
-	Soustraction
*****	Multiplication
/	Division entière
%	Reste de la division entière

IV.2 Les opérateurs d'affectation

Ces opérateurs permettent d'affecter à une variable une valeur, certains de ces opérateurs permettent d'affecter et de réaliser une opération arithmétique en même temps.

Opérateur	Fonction	Ecriture	Equivalence
=	Affectation ordinaire	X=Y	X=Y
+=	Affectation d'une addition	X+=Y	X=X+Y
-=	Affectation d'une soustraction	X-=Y	X=X-Y
***=**	Affectation d'une multiplication	X*=Y	X=X*Y
/=	Affectation d'une division entière	X/=Y	X=X/Y
%=	Affectation du reste de la division entière	X%=Y	X=X%Y
--	Décrémentation de 1	X--	X=X-1
++	Incrémentation de 1	X++	X=X+1

IV.3 Les opérateurs logiques bit à bit

Ces opérateurs agissent sur des mots binaires. Ils effectuent entre deux mots une opération logique sur les bits de même rang.

Opérateur	Fonction	Notation
&	ET	Z=X&Y
|	OU	Z=X|Y
^	OU exclusif	Z=X^Y
~	NON	Z=~Y
>>	Décalage à droite des bits	Z=X>>4 (Z prend la valeur de X après le décalage à droite de 4 bits)
<<	Décalage à gauche des bits	Z=X<<4 (Z prend la valeur de X après le décalage à gauche de 4 bits)

IV.4 Les opérateurs de tests conditionnels

Ces opérateurs s'adressent uniquement aux opérations de test conditionnel. Le résultat de ces tests est binaire : vrai(=« 1 ») ou faux(= « 0 »).

Les opérateurs logiques :

Opérateur	Fonction
&&	ET logique
||	OU logique
!	NON logique

Les opérateurs de comparaison :

Ces opérateurs renvoient la valeur « 0 » si la condition vérifiée est fausse, et « 1 » si la condition vérifiée est vraie.

Opérateur	Fonction
==	Egal à
!=	Différent de
>	Supérieur à
<	Inférieur à
>=	Supérieur ou égal à
<=	Inférieur ou égal à

V Les structures du langage C

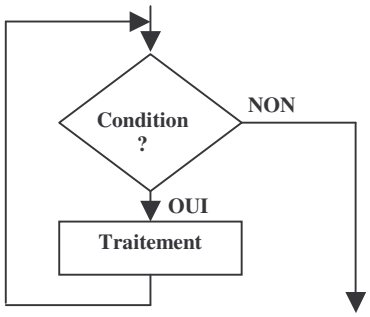
V.1 Les structures répétitives (ou itératives)

V.1.1 Structure « while » : tant que … faire …

```
while (condition)
        {
                traitement ;
        }
```

Dans cette structure, la condition est testée au début.

Equivalence en algorigramme :

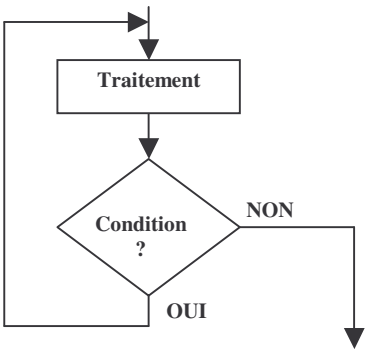


V.1.2 Structure « do … while » : faire … tant que

```
do
        {
                traitement ;
        }
        while (condition) ;
```

Dans cette structure, la condition est testée à la fin.

Equivalence en algorigramme :



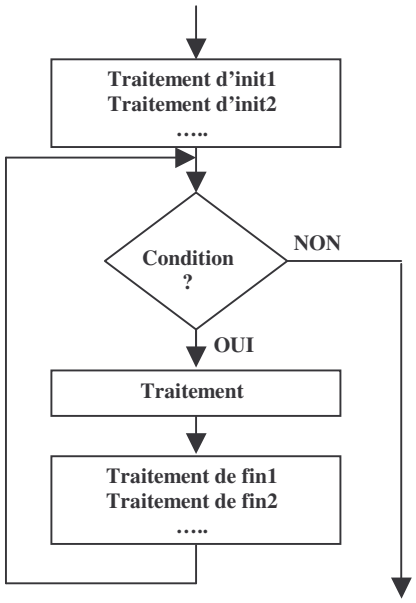
V.1.3 Structure « for » : pour…faire…jusqu'à…

```
for(traitement d'init1, traitement
d'init2,… ; condition ; traitement de
fin1, traitement de fin2,…)
        {
                traitement ;
        }
```

Cette structure se déroule en quatres parties :

-traitement d'initialisation,
-test de la condition,
-traitement,
-traitement de fin.

Equivalence en algorigramme :



V.2 Les structures conditionnelles (ou alternatives)

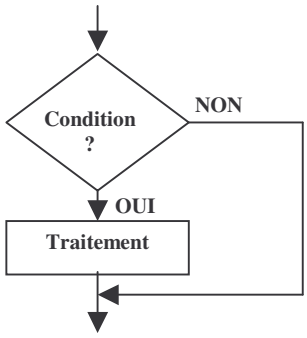
Ces structures permettent d'exécuter des séquences différentes en fonction d'une condition.

V.2.1 Structure « if » : si…faire

```
if(condition)
        {
                traitement ;
        }
```

On exécute le traitement uniquement si la condition est réalisée.

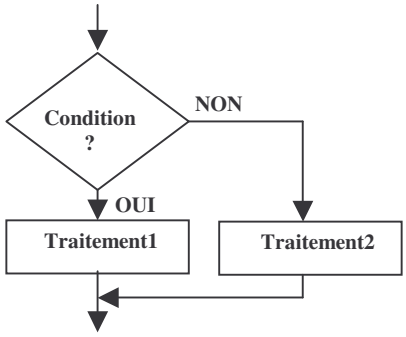
Equivalence en algorigramme :



V.2.2 Structure « if…else » : si faire…sinon faire

```
if(condition)
        {
                traitement ;
        }
else
        {
                traitement2 ;
        }
```

Equivalence en algorigramme :

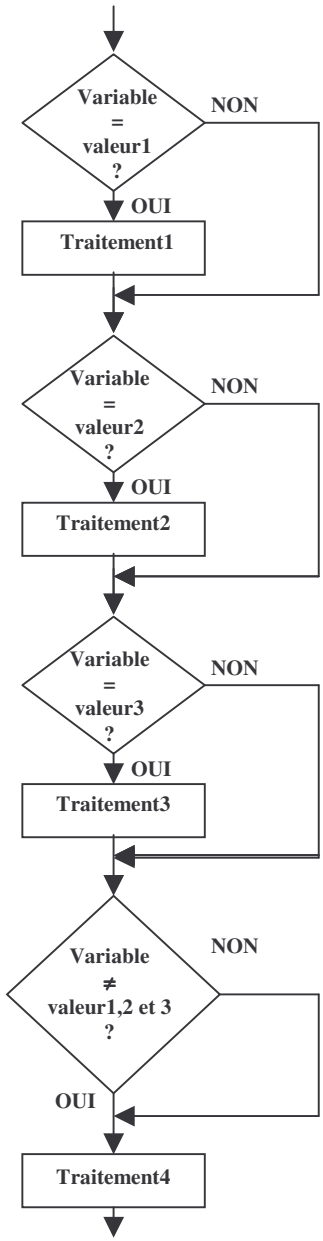


V.2.3 Structure de choix « switch…case »

```
switch(variable)
        case 'valeur1' :
                {
                        traitement1 ;
                } ;
        case 'valeur2' :
                {
                        traitement2 ;
                } ;
        case 'valeur3' :
                {
                        traitement3 ;
                }
        default :
                {
                        traitement4 ;
                }
```

Permet d'exécuter des traitements différents en fonction de la valeur d'une variable.

Equivalence en algorigramme :



VI La fonction principale d'un programme C (programme principal)

La syntaxe pour déclarer la fonction principale d'un rpogramme C (programme principal) est la suivante :

```
void    main(void)
        {
        …
        …
        }
```

VI Les fonctions (sous programmes)

Afin de réduire la taille de la fonction principale de programme C il est souvent préférable de définir sous forme de fonctions une même suite d'instructions appelées plus d'une fois dans le programme principal.

Les fonctions du langage "C" peuvent renvoyer des valeurs de même qu'elles peuvent prendre en compte des arguments provenant de la procédure d'appel.

S'il n'y pas de renvoi ou aucun argument, on saisit le mot clé « **void** » en remplacement.

Exemple :

```
//déclaration de la fonction

void    fonction1(void)
        {
        …
        …
        }

//utilisation de la fonction
        …
        fonction1() ;
        …
```

La valeur renvoyée est définie après le mot clé « **return** ».

Lorsque l'on veut, dans une fonction, modifier une variable passée en argument il est obligatoire d'utiliser un pointeur.

Ce document a été réalisé à partir de deux cours :

-« Le langage C adapté aux microcontrôleur » de Jean Luc PADIOLLEAU,

-« Introduction au langage C » de Philippe LETENNEUR et Philippe LECARDONNEL.