

LE LANGAGE "C" ADAPTÉ AUX MICROCONTRÔLEURS



1. PRÉSENTATION.

Le langage "C" a fait son apparition en 1972 pour le développement du système d'exploitation Unix. Il est devenu un standard de la norme ANSI en 1983. Lui, ainsi que son petit frère le C++, sont très utilisés pour le développement d'applications sous station Unix et PC.

Depuis quelques années il a fait son entrée dans le monde des microcontrôleurs. Il permet de bénéficier d'un langage universel et portable pratiquement indépendant du processeur utilisé. Il évite les tâches d'écritures pénibles en langage assembleur et élimine ainsi certaines sources d'erreurs.

2. ARCHITECTURE D'UN PROGRAMME C POUR µC.

La saisie d'un programme en "C" répond pratiquement toujours à la même architecture. On peut noter que le symbole "#" est suivi d'une directive de compilation, le symbole "//" est suivi d'un commentaire.

```
#include <stdio.h> // Directive de compilation indiquant d'inclure la bibliothèque E/S standard
#include <reg_uc.h> // Directive de compilation indiquant d'inclure la bibliothèque spécifique au µC

#define clear=0x00 // Directive de compilation indiquant des équivalences ...
char val1=0xA5; // Déclaration d' une variable " caractère" avec valeur initiale
int val2; // Déclaration d' une variable " nombre entier" ...
void tempo(char temps) // Fonctions et procédures appelées plusieurs fois dans le programme principal
{ ... }
void main(void) // Programme principal
{
DDRBA=0xFF // initialisation et configuration
....
while (1) // Boucle principale
{
... tempo(100); ...
val2=bintobcd(val1); ...
}
}
```

Chaque ligne d'instruction se termine par un ";" .

Le début d'une séquence est précédé du symbole "{" .

La fin d'une séquence est suivie du symbole "}" .

La notation des nombres peut se faire en décimal de façon normale ou en hexadécimal avec le préfixe "0x" .

3. LES TYPES DE DONNÉES DU LANGAGE "C".

Il existe, dans le langage "C", plusieurs "types" de données classés selon leurs tailles et leurs représentations. On ne détaillera que ceux utilisés dans le cadre des microcontrôleurs.

3.1 TYPES DE BASE.

On en rencontre généralement trois types qui peuvent être signés ou non signés. Dans ce dernier cas la déclaration sera précédée du mot clé "unsigned"

TYPE	DÉFINITION	TAILLE	DOMAINE NON SIGNÉ	DOMAINE SIGNÉ
char	Variable de type caractère ascii mais utilisée pour les nombres entiers	8 bits	0 à 255	-128 à +127
int	Variable de type nombre entier	16 bits	0 à 65536	-32768 à 32767
float	Variable de type nombre réel	32 bits	+/- 3,4.10 ⁻³⁸ à 3,4.10 ³⁸	X

Remarques : Dans la déclaration de certaines fonctions, on emploie le type “ void” qui signifie que la fonction ne renvoie ou n’ exige aucune valeur.

3.2 TYPES STRUCTURÉS.

Les types structurés sont en fait une association de plusieurs variables de base de même type. Il en existe deux types :

- Les types **tableau** dont la taille est définie - ex : `int tableau[10]` ; *tableau de 10 entiers.*
- Les types **pointeurs** dont la taille n’est pas définie - ex : `char *chaine` ; *pointeur de caractères.*

Les chaînes de caractères peuvent être définies par les 2 types. On préfère cependant le pointeur pour sa taille indéfinie.

Remarque :

- Le pointeur est comparable à un registre d’index qui contient non pas une donnée mais l’adresse de cette données.
- L’adresse d’une variable, affectant un pointeur, s’obtient en précédant son nom du symbole “&”.
- Une chaîne de caractère se termine toujours par le caractère nul “\0”.

4. LES OPÉRATEURS.

4.1 LES OPÉRATEURS ARITHMÉTIQUES.

Ces opérateurs permettent d’effectuer les opérations arithmétiques traditionnelles : Addition, soustraction, multiplication et division entière.

OPÉRATEUR	FONCTION
+	Addition
-	Soustraction
*	Multiplication
/	Division entière
%	Reste de la division entière

4.2 LES OPÉRATEURS D’AFFECTATION.

L’opérateur indispensable au langage “ C” est l’ affectation défini principalement par le signe “=”. Il permet de charger une variable avec la valeur définie par une constante ou par une autre variable. Il en existe d’ autres qui, en plus de l’ affectation, effectue une opération arithmétique.

OPÉRATEUR	FONCTION	EQUIVALENCE
=	Affectation ordinaire	X=Y
+=	Additionner de _	X+=Y équivalent à X=X+Y
-=	Soustraire de _	X-=Y équivalent à X=X-Y
=	Multiplier par _	X=Y équivalent à X=X*Y
/=	Diviser par _	X/=Y équivalent à X=X/Y
%=	Modulo	X%=Y équivalent à X=X%Y
--	Soustraire de 1 (Décrément)	X-- équivalent à X=X-1
++	Ajouter 1 (Incrément)	X++ équivalent à X=X+1



4.3 LES OPÉRATEURS LOGIQUES.

Ces opérateurs s'adresse uniquement aux opérations de test conditionnel. Le résultat de ces opérations est binaire : "0" ou "1".

OPÉRATEUR	FONCTION
&&	ET logique
	OU logique
!	NON logique

4.4 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
&	ET
	OU
^	OU exclusif
~	NON
>>	Décalage à droite des bits
<<	Décalage à gauche des bits

4.5 OPÉRATEURS DE COMPARAISON.

Ces opérateurs renvoient la valeur " 0" si la condition vérifiée est fausse, sinon ils renvoient " 1" .

OPÉRATEUR	FONCTION
==	Égale à
!=	Différent de
>	Supérieur à
<	Inférieur à
>=	Supérieur ou égal à
<=	Inférieur ou égal

5. LES STRUCTURES RÉPÉTITIVES.

Le langage "C" possède des instructions permettant de répéter plusieurs fois une même séquence en fonction de certaines conditions.

5.1 STRUCTURE "WHILE" : TANT QUE ... FAIRE ...

Avec ce type d'instruction le nombre de répétitions n'est pas défini et dépend du résultat du test effectué sur la condition. Si cette dernière n'est jamais vérifiée, la séquence n'est pas exécutée.

```
while (int x!=0)
{
}
```

La structure précédente répète la suite d'instruction comprises entre crochets tant que la variable entière "x" est différente de 0.



5.2 STRUCTURE “DO ... WHILE” : FAIRE ... TANT QUE...

Cette structure ressemble fortement à la précédente à la seule différence que la séquence à répéter est au moins exécuter une fois même si la condition n'est jamais vérifiée.

```
Do {  
    ...  
} while (int x!=0);
```

5.3 STRUCTURE “FOR” : POUR <VARIABLE> ALLANT DE <VALEUR INITIALE> À <VALEUR FINALE> FAIRE...

Cette instruction permet de répéter, un nombre de fois déterminé, une même séquence.

```
for (i=0;i<5;i++)  
{  
    ...  
}
```

La structure précédente répète 5 fois la suite d'instruction comprise entre crochets. La variable “i” prendra les valeurs successives de : 0, 1, 2, 3 et 4.

6. LES STRUCTURES ALTERNATIVES.

Ces structures permettent d'exécuter des séquences différentes en fonction de certaines conditions.

6.1 STRUCTURE “IF ... ELSE” : SI <CONDITION> FAIRE ... SINON FAIRE ...

Avec cette structure on peut réaliser deux séquences différentes en fonction du résultat du test sur une condition

```
if (a<b) c=b-a;  
else c=a-b;
```

La structure précédente affecte la valeur “b-a” à “c” si “a” est inférieur à “b” sinon “c” est affecté par la valeur “a-b”.

6.2 STRUCTURE “SWITCH ... CASE”.

Cette structure remplace une suite de “if ... else if ...else” et permet une de réaliser différentes séquences appropriées à la valeur de la variable testée.

```
switch (a)  
{  
    case '1' : b=16;  
    case '2' : b=8;  
    case '3' : b=4;  
    case '4' : b=2;  
}
```

Dans la structure précédente “b=16” si “a=1”, “b=8” si “a=2” etc.

7. LES FONCTIONS.

Afin de réduire la taille du programme il est souvent préférable de définir sous forme de fonction une même suite d'instructions appelée plus d'une fois dans le programme.

La fonction principale d'un programme “C” est définie grâce au mot clé “main”.

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. Si il n'y pas de renvoi ou aucun argument, on saisit le mot clé "void" en remplacement.

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. Une fonction doit toujours être définie avant sa procédure d'appel. Dans le cas contraire une simple déclaration doit être faite dans l'en tête du programme.

Exemples :

Fonction permettant de positionner à 1 le bit de rang "n" d'un registre 8 bits "reg".

```

void setbit(char *reg,char n)
{
char m=1<<n; // variable m = 1 décalé de n fois à gauche
*reg=*reg | m; // opération bit à bit OU entre *reg et m
}

void main(void); // Programme principal
{
...
While(1) // boucle principale
{
...
setbit(&DRB,5); // mise à 1 du bit 5 de DRB
... }

```

8. Synthèse

Fonctions	Opérateur	Description	Exemples
Identificateurs	()	Appel de fonction	
	[]	Indice de tableau	tableau[3]=5;
opérateurs unaires	!	Négation logique (NOT)	b=!a;
	~	Complément binaire (1)	b=~a
	-	Moins unaire	b=-a;
	+	Plus unaire	b+=a;
	++	Préincrément ou postincrément	b=a++; (b=a puis a=a+1)
	--	Prédécrément or postdécrément	b=a--; (b=a puis a=a-1)
	&	Adresse de	b=&a; (b égale l'adresse de a)
	*	Indirection (adressage indexé)	b=*a; (b=contenu de l'adresse de a)
opérateurs binaires	*	Multiplication	c=a*b;
	/	Division	c=a/b;
	+	Plus binaire	c=a+b;
	-	Moins binaire	c=a-b;
	<<	Décalé à gauche	c=a<<b; (a est décalé b fois à gauche)
	>>	Décalé à droite	c=a>>b; (a est décalé b fois à droite)
	&	ET entre bits	c= a & b; (ET logique bit à bit)
	^	OU exclusif entre bits	c= a ^b;
		OU entre bits	c= a b;
Tests	<	Strictement inférieur	if a < b
	<=	Inférieur ou égal	if a >= b
	>	Strictement supérieur	if a > b
	>=	Supérieur ou égal	if a >= b
	==	Egal	if a ==b (si a est égale à b)
	!=	Différent	if a != b
	&&	ET logique	if ((a=5) && (b=2))
		OU logique	if ((a=5) (b=2))
	?:	condition	z=(a>b)?a:b (Si a>b a z=a sinon z=b)
Affectation	=	Affectation simple	a=b; (a prend la valeur de b)
Auto-affectations	*=	Affectation produit	a*=2 (a=a*2)
	/=	Affectation quotient	a/=2 (a= a/2)
	%=	Affectation reste	a%=2 (a= le reste de a/2)
	+=	Affectation somme	a+=2 (a=a+2)
	-=	Affectation différence	a-=2 (a=a-2)
	&=	Affectation ET entre bits	a&=5 (a=a&5)
	^=	Affectation OU EX entre bits	a^=5 (a=a^5)
	=	Affectation OU entre bits	a =5 (a=a 5)
	<<=	Affectation décalé gauche	a<<=5 (a=a<<5)
	>>=	Affectation décalé droite	a>>=5 (a=a>>5)



9. Algorithmie

1. Qu'est ce qu'un Algorithme ?

Un Algorithme est une suite ordonnée de directives composées d'actions et de décisions qu'il faut exécuter en séquence suivant un enchaînement strict pour accomplir une tâche quelconque. Il ne tient pas compte des technologies mises en œuvre (indépendant d'un langage informatique).

2. Qu'est ce qu'un Algorithme : ?

Il reproduit dans un langage graphique normalisée tous les cheminements du raisonnement logique qui détermine la composition de l'Algorithme.

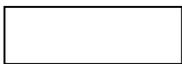
3. Symbolisation :

Symboles

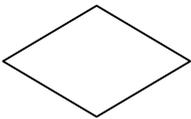
Désignation



Début ou Fin d'un Algorithme



Action, Traitement



Test, Décision



Sous programme :
petite partie d'un programme pouvant être appelé régulièrement.

On s'évite de le réécrire systématiquement.



Entrée ou sortie de variables.
Exemple : Affichage de valeurs sur écran.



Opération manuelle.
Préparation. Exemple : **Initialisation** de périphériques.

4. Réaliser l'algorithme.

1° Utilisation de votre machine à calculer :

- $x = a \cdot b + c$
- $x = (a - b) \cdot c$
- $x = (a + b) / (a + d)$

2° Gestion d'un feu de carrefour simple (sans détecteur).

3° Comparaison de trois nombres entrées au clavier, et on affiche le plus grand, le moyen et le plus petit.

4° On entre 10 chiffres et on affiche le plus grand (MAX) et le plus petit (MIN).

