AIDE MÉMOIRE C POUR DSPIC

Architecture du fichier source

```
/*****************************
*   Titre et/ou commentaires *
*****************************/

#include  <p30f2010.h>

//Variables
int  CONSIGNE, POSITION; //Description
char IND_PAS;            // Description

//Constantes mémorisées en ROM
const char TAB_PAS[] = {8,2,4,1};
const int DEL_MAX = 1000;

//Constantes non mémorisées
#define NB1    0x1234;
#define PHRASE "Autre chaîne";

//Fonctions
int MA_FONCTION(int A,int B)
 {
  int LOCAL;
  LOCAL=A*B;
  return LOCAL;
 }

//Programmes d'interruption
// (exemple sur "Timer 1")
#void _ISR _T1Interrupt (void)
 {
  int ERREUR;
  ERREUR=MA_FONCTION(CONSIGNE,POSITION);
  … … …
  //Acquitement interruption P2.2
  IFS0bits.T1IF=0;
 }

//Programme principal
int main(void)
 {
  int i;
  //Initialisations des variables
  CONSIGNE = 0x8000;
  POSITION = 0x8000;
  //Initialisation des registres
  TRISC=0x9FFF; // RC14 et RC13 en sortie
  IEC0bits.T1IE=1;//Valid. int. Timer 1

  while (1)
   {
    … …
   }
 }
```

Tout le texte délimité par "/*" et "*/" est ignoré par le compilateur.
De même pour les caractères d'une ligne qui suivent un "//".

Fichier inclus de type "entête" (header). Défini les constantes spécifiques au µC utilisé (adresses des registres, etc.)

Déclaration des variables en RAM :
- `int` : integer = entier (taille 16 bits)
- `char` : octet non signé (taille 8 bits)

Affectation de constantes en ROM (ici une table de 4 octets et un entier sur 16 bits)

Définitions de constantes non mémorisées

Déclaration et définition des fonctions (ou sous-programmes)
Les paramètres d'entrée et de sortie sont optionnels (utiliser alors "`void`")
Les accolades (`{}`)encadrent le corps de la fonction
Déclaration d'une variable locale
Appel de la fonction

Déclaration et définition d'une fonction d'interruption (ici sur le dépassement du compteur du "Timer 1").

Variables locales éventuelles
Ne pas oublier d'acquiter l'interruption

Fonction principale lancée au "reset" :
- Déclaration de variables locales
- Partie initialisations des variables
- Partie initialisation des registres du µC
- Boucle sans fin qui active successivement les différentes fonctions logicielles principales

L'identificateur "`main`" est obligatoire et la fonction ne comporte aucun paramètre.

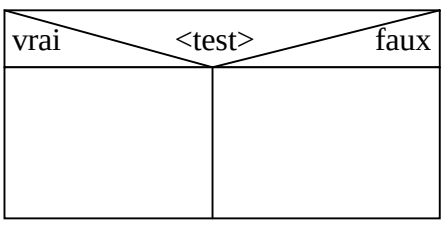
Éléments de syntaxe

- Les **majuscules** et **minuscules** sont distinguées. Utiliser de préférence les majuscules pour les identificateurs personnels, sauf si la lisibilité en est réduite.
- **Déclaration de variables et constantes** :
  - Le type (la nature) de la variable ou de la constante précède son identificateur. Exemples :
    - `int NOMBRE1, NOMBRE2;` : 2 entiers signés sur 16 bits (code C2 : de –32768 à 32767)
    - `unsigned int VAR1;` : entiers non signés sur 16 bits (de 0 à 65535)
    - `char TAB[10];` : tableau de 10 octets non signés (valeurs de chaque octet : 0 à 255)
    - `const char TAB_CHAR[]={'A','B','C'};` : caractères codés en ASCII sur 8 bits
    - `const char CHAINE[]="ABC";` : chaîne de caractères. Déclaration identique à `TAB_CHAR`, mais le compilateur ajoute un 0 à fin de la table.
  - Variables "globales" reconnues par l'ensemble du programme : elles sont déclarées à l'extérieur de toute fonction (y compris "main").
  - Variables "locales" : elles sont déclarées au début du bloc d'une fonction. La variable n'est pas reconnue dans les autres fonctions.
    Dans certains cas (fonctions simples), le compilateur peut décider de placer certaines variables dans des registres du CPU pour accélérer la vitesse de traitement
- Les lignes de déclaration et d'instruction se terminent par "`;`"
- Un "bloc" est un ensemble de lignes de déclarations et d'instructions encadrées par "`{`" et "`}`"
- **Déclaration d'une fonction**. Une fonction traite des données pour obtenir un résultat.
  - Les types et les identificateurs des données sont déclarés entre les parenthèses et séparés par des virgules (ex : `MA_FONCTION(int A,int B)` utilise les variables A et B dans son traitement)
  - Le type du résultat correspond au type de la fonction (ex : `int MA_FONCTION` renvoit un résultat "entier signé" sur 16 bits)
  - Le résultat est retourné avec l'instruction "**`return`**" (voir `MA_FONCTION` et son appel sur la 1° page) qui termine aussi l'exécution de la fonction
  - La déclaration d'une fonction de traitement d'interruption doit être précédée de la directive `_ISR`. Son nom ne peut être modifié (voir le même exemple)
  - La fonction principale lancée au reset est particulière, elle n'a aucun paramètre et se nomme obligatoirement "`main`"
- **Structures conditionnelles :**

Sélection binaire

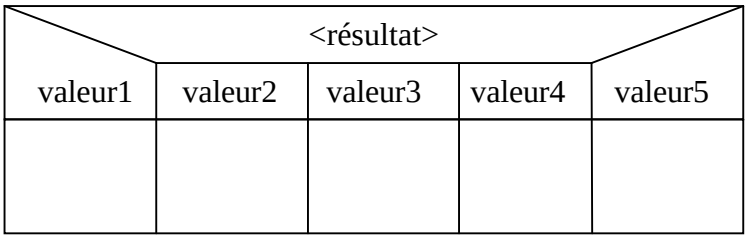
```
if (<test>)
 {
  <traitement si vrai>
 }
else
 {
  <traitement si faux>
 }
```

Le bloc "else" est optionnel



Sélection multiples

```
switch (<résulat>)
 {
  case <valeur1>:
   {…}
  break;
  case <valeur2>:
   {…}
  break;
  default:
   {…}
  break;
 }
```



➢ **Structures itératives**

Nb d'itérations déterminé

```
for (exp1;exp2;exp3)
 {
  …
 }
```

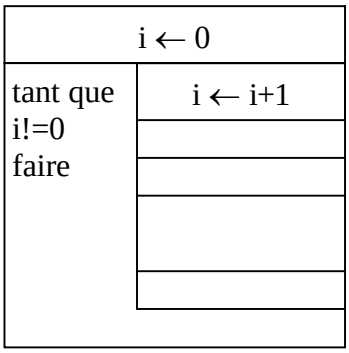
Condition au début

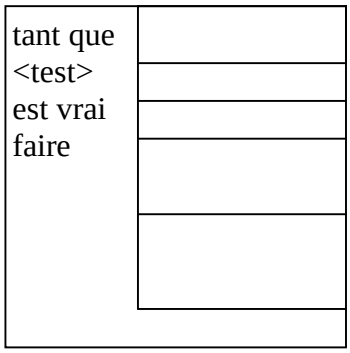
```
while (<test>)
 {
  …
 }
```

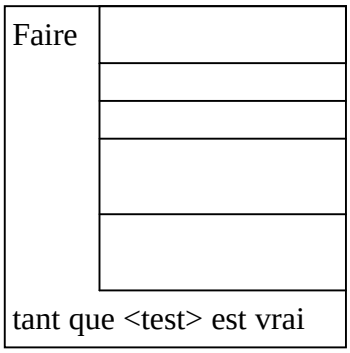
Condition à la fin

```
do
 {
  …
 }
while (<test>)
```

Graphes NS correspondants :







`<exp1>` de type : i=0 : valeur initiale du compteur d'itérations
`<exp2>` de type : i!=0 : condition d'itération
`<exp1>` de type : i++ : modification du compteur d'itérations
`<test>` est le résultat binaire d'un opérateur relationnel (vrai ou faux). Exemples : `"A==B"` `"A>B"`

➢ **Opérateurs de calculs :**

Opération	Syntaxe	Exemples
Affectation	=	A=3; A=A+B+C;
Addition	+	A=B+C;
Soustraction	-	A=B-C;
Multiplication	*	A=B*C;
Division	/	A=B/C;
ET logique	&	X=Y&Z;
OU logique	|	X=Y|Z;
XOR logique	^	X=Y^Z;
Inversion logique	!	X=!Y;
Décalage à G	<<	X=Y<<2;
Décalage à D	>>	X=Y>>4;
Complément log.	~	X=~Y;

Cas des opérations logiques :
Elles se font "bit à bit" comme l'illustre l'exemple ci-dessous :

```
R = A & B;
```

Variables de type "char" (8 bits)
A=155=0x9B et B=120=0x78

Bit	7	6	5	4	3	2	1	0
A	1	0	0	1	1	0	1	1
B	0	1	1	1	1	0	0	0
R	0	0	0	1	1	0	0	0

Le résultat donne R=0x18=24

Cas particuliers :

Opération	Syntaxe	Exemples	Opération équivalente
ET et affectation	&=	A&=B;	A=A&B;
OU et affectation	|=	A|=B;	A=A|B;
XOR et affect.	^=	A^=B;	A=A^B;
Incrémentation	++	I++;	I=I+1;A=I+I++;
Décrémentation	--	--I;	I=I-1;A=I---I;

➢ **Opérateurs relationnels :**

Opération	Syntaxe	Exemples
Egal à	==	if (A==B)
Différent de	!=	if (A!=B)
Inférieur à	<	if (A<B)
Supérieur à	>	if (A>B)
Inférieur ou égal	<=	if (A<=B)
Supérieur ou égal	>=	if (A>=B)

Opération	Syntaxe	Exemples
OU	||	if ((A==B)||(A==C))
ET	&&	if ((A==B)&&(A==C))