La programmation des PIC en C

Les expressions, les alternatives, les itérations

Réalisation : HOLLARD Hervé.
http://electronique-facile.com
Date : 26 aout 2004
Révision : 1.4

Sommaire

Introduction

Les microcontrôleurs PIC de la société Microchip, sont depuis quelques années dans le "hit parade" des meilleures ventes. Ceci est en partie dû à leur prix très bas, leur simplicité de programmation, les outils de développement que l'on trouve sur le NET.

Aujourd'hui, développer une application avec un PIC n'a rien d'extraordinaire, et tous les outils nécessaires sont disponibles gratuitement. Voici l'ensemble des matériels qui me semblent les mieux adaptés.

Ensemble de développement (éditeur, compilateur, simulateur) :
MPLAB de MICROCHIP **http://www.microchip.com**
Logiciel de programmation des composants:
IC-PROG de Bonny Gijzen **http://www.ic-prog.com**
Programmateur de composants:
PROPIC2 d'Octavio Noguera voir notre site **http://electronique-facile.com**

Pour la programmation en assembleur, beaucoup de routines sont déjà écrites, des didacticiels très complets et gratuits sont disponibles comme par exemple les cours de **BIGONOFF** dont le site est à l'adresse suivante **http://abcelectronique.com/bigonoff**.

Les fonctions que nous demandons de réaliser à nos PIC sont de plus en plus complexes, les programmes pour les réaliser demandent de plus en plus de mémoires. L'utilisateur est ainsi à la recherche de langages "évolués" pouvant simplifier la tâche de programmation.

Depuis l'ouverture du site **http://electronique-facile.com**, le nombre de questions sur la programmation des PIC en C est en constante augmentation. Il est vrai que rien n'est aujourd'hui disponible en français.

Mon expérience dans le domaine de la programmation en C due en partie à mon métier d'enseignant, et à ma passion pour les PIC, m'a naturellement amené à l'écriture de ce **didacticiel**. Celui-ci se veut **accessible à tous** ceux qui possèdent une petite expérience informatique et électronique (utilisation de Windows, connaissances minimales sur les notions suivantes : la tension, le courant, les résistances, les LEDs, les quartz, l'écriture sous forme binaire et hexadécimale.).

Ce troisième fascicule vous permettra enfin de réaliser vos premier programmes. Vous serez capable de rendre vos applications "intelligentes" si celle-ci ne demande pas de gestion du temps ou de calculs complexes.

Structure de ce document

Ce document est composé de chapitres. Chaque chapitre dépend des précédents. Si vous n'avez pas de notion de programmation, vous devez réaliser chaque page pour progresser rapidement.

Le type gras sert à faire ressortir les termes importants.

Vous trouverez la définition de chaque **terme nouveau** en **bas de la page** où apparaît pour la première fois ce terme. *Le terme est alors en **italique**.*

La couleur **bleue** est utilisée pour vous indiquer que ce **texte** est **à taper** exactement sous cette forme.

La couleur **rouge** indique des **commandes informatiques** à utiliser.

Le matériel nécessaire

Les deux logiciels utilisés lors du premier fascicule.

Un programmateur de PIC comprenant un logiciel et une carte de programmation. Vous trouverez tout ceci sur notre site.

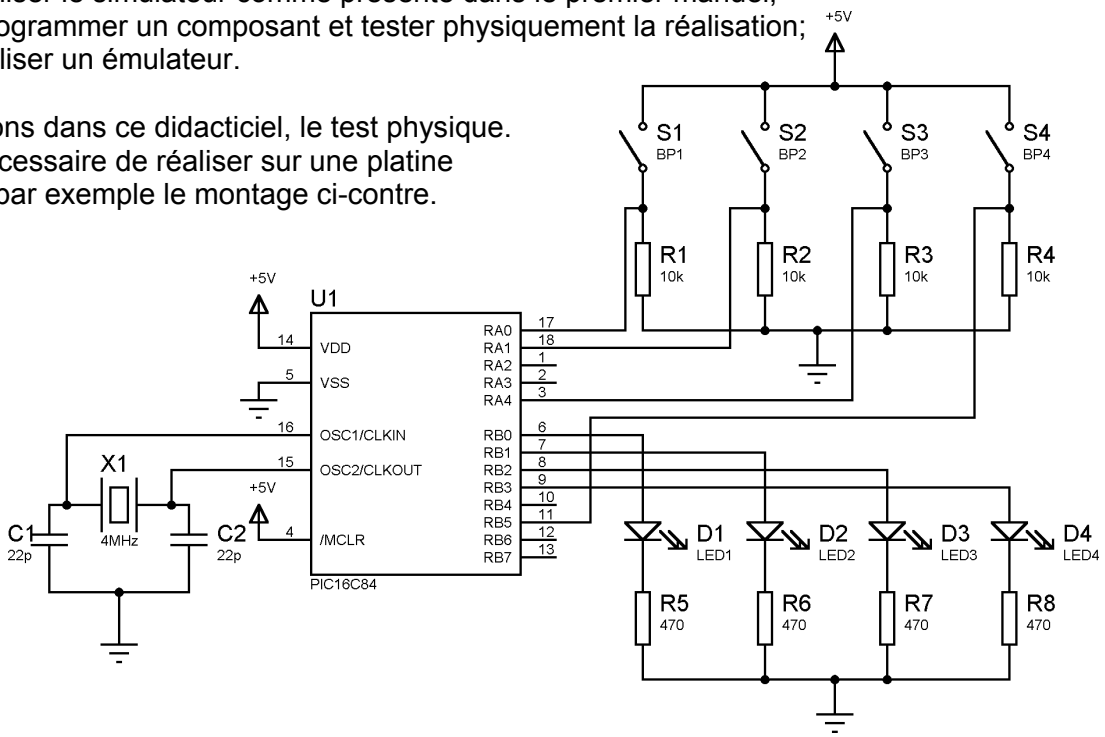
Un PIC 16F84, un quartz de 4MHz, deux condensateurs de 22pF, 4 leds rouges, 4 résistances de 470 ohms, 4 interrupteurs, 4 résistances de 10 Kohms. Une platine d'essai sous 5 volts.

La platine d'essai

Pour tester les programmes proposés il est possible :

-utiliser le simulateur comme présenté dans le premier manuel;
-programmer un composant et tester physiquement la réalisation;
-utiliser un émulateur.

Nous utiliserons dans ce didacticiel, le test physique. Il est ainsi nécessaire de réaliser sur une platine de type LAB par exemple le montage ci-contre.



But à atteindre

Ce didacticiel vous permettra de **faire des choix, des boucles**, ainsi que d'utiliser les **principales expressions du C**. Vous apprendrez aussi à utiliser **les variables** afin de vous servir de la mémoire du microcontrôleur.

Pour atteindre ces buts, nous allons réaliser un programme de gestion d'éclairage constitué de 4 boutons poussoirs et de 4 leds (Voir page 4). Chaque bouton poussoir pourra réaliser des actions déterminées sur les leds.

Les variables

Une variable est **une portion réservée de la mémoire RAM** à laquelle on a donné un nom. Elle est utilisée afin de garder en mémoire une entité acquise dans le temps, pour la réutiliser plus tard.

Toute variable en C possède un type. Pour la suite de ce didacticiel nous n'utiliserons que deux types de variables le "char"et le "bit".

- **Le type char** occupe **un octet en mémoire.**
  Nous pouvons déposer dans cette variable un nombre entre 0 et 255.
- **Le type bit** occupe **un bit en mémoire.**
  Nous pouvons déposer dans cette variable un 0 ou un 1 logique.

La déclaration d'une variable se fait de deux façons:

- **"type" "nom" @ "adresse-de-la-portion_reservé"**
- **"type" "nom"** (le compilateur réserve une portion encore libre)

La place de la déclaration des variables dans le programme est importante. Pour ce didacticiel, nous utiliserons des variables permanentes (leur place est réservée durant tout le programme), nous les déclarerons au début du fichier.

Il est important de connaître la place des variables dans la mémoire pour les visualiser lors de la simulation. Il est aussi indispensable de ne pas créer plus de variables que de mémoire disponible.
Pour cela, lors de la compilation, une table récapitulative de l'état de la mémoire RAM est affichée.

Voici un petit programme, qui ne fait rien, mais sert uniquement à comprendre le processus.

```
// Attention de respecter les majuscules et minuscules

//-------------------------declaration des variables----------------------------

char a @ 0x11;          // reservation d'un octet nomme a à l'adresse 0x11
bit b @ 0x12.2;         // reservation d'un bit nomme b à la place 2 de l'octet 0x012
char c, d;              // reservation de deux octets nommes c et d
bit e;                  // reservation d'un bit nomme e

//-----------------------------Fonction principale-----------------------------

void main(void)
{
   nop();               // instruction qui ne fait rien
}
```

Taper ce programme, le compiler. Les messages d'information de la compilation nous informe de l'état de la RAM, nous allons étudier ces messages.

```
RAM:  00h  : -------- ----..7* *-7***** ********
RAM:  20h  : ******** ******** ******** ********
RAM:  40h  : ******** ********
RAM usage: 5 bytes (0 local), 63 bytes free
```

Explication des lignes 1, 2 et 3 :

- Chaque symbole représente l'état d'un octet. La première ligne, par exemple, récapitule l'état des octets 00h à 19h (le h signifie que nous parlons en hexadécimal).
- \- Octet réservé par le compilateur pour le fonctionnement du microcontrôleur (registre PORTA, TRISB, …) ou par l'utilisateur à une place précise (0X11).
- . Octet réservé par l'utilisateur, mais dont la place n'a pas d'importance (variable c et d)
- 7 Octet où 7 bits sont disponibles.
- * Octet libre.

La ligne 4 récapitule l'état de la RAM.

Il est possible de modifier l'exemple final du fascicule précédent en modifiant les #define par des variables. En effet, les entrées et les sorties sont représentées par deux octets (registres) de la mémoire : PORTA et PORTB .

Le programme suivant est identique à celui du fascicule 2.

```
// Attention de respecter les majuscules et minuscules
//--------------------------------E/S-----------------------------------------------
char sortie @ PORTB;
bit inter1 @ RA0;
bit inter2 @ RA1;
bit inter3 @ RA4;
bit inter4 @ RB5;
bit led1 @ RB0;
bit led2 @ RB1;
bit led3 @ RB2;
bit led4 @ RB3;
//------------------------------Fonction principale----------------------------
void main(void)
{
   sortie = 0;                    // Initialisation des pattes du microcontroleur
   TRISB = 0b11110000;

   for (;;) {                     // La suite du programme s'effectue en boucle
              led1 = inter1;
              led2 = inter2;
              led3 = inter3;
              led4 = inter4;
            }
}
```

Les expressions

Ce chapitre est tiré de l'aide au langage C du logiciel DevPic84C.

Les **expressions en C vont nous permettre de réaliser des opérations entre plusieurs entités**. Une expression est composée d'un *opérateur*[1] et de un ou plusieurs *opérandes*[2].
Nous allons distinguer 5 types d'expressions :

- I / les affectations;
- II / les opérations unaires;
- III / les opérateurs binaires
- IV / les auto-affectations;
- V / les comparaisons.

I / Les affectations

- **" = " Affectation** :

  Place la valeur du 2e opérande dans le 1er.
  Le premier opérande ne peut pas être une constante.
  Exemple

```
a = 5;          // place la valeur 5 dans a
```

II / Les opérations unaires

Elles n'admettent qu'un seul opérande.

- **" . " Sélection d'un bit** (le point) :

  Fait référence à un bit de l'opérande.
  Exemple :

```
PORTA.2 = 1;    //  met RA2 à 1
```

- **" - " Négation** :

  Change le signe de l'opérande.
  Exemple :

```
a = 10;
b = - a;        // b = - 10
```

- **" ! " Complémentation logique** :

  Complémente une variable de type bit.
  Exemple :

```
bit a,b;
a = 1;
b = !a;         // b = 0
```

- **" ~ " Complémentation à un** :

  Inverse tous les bits de l'opérande.
  Exemple :

```
char a, b ;
a  =  5;    //  a = 0000 0101  soit  5
b  = ~a;    //  b = 1111 1010  soit 255 - 5
```

[1] Symbole représentant une opération. Ex: =, + …
[2] Entité sur laquelle porte l'opération. Cette entité peut être fausse si elle est égale à 0, vraie dans les autres cas.

" ++ " Incrémentation :

Ajoute 1 à l'opérande.

```
Exemples :   a = 14;
             a++;            // après exécution a=15
             c = ++a;        // pré-incrémentation : a = (a+1) puis c = a
                             // après exécution c = 16
```

Attention : l'instruction ci-dessous est impossible avec le compilateur CC5X.

```
             d = b++;        // post-incrémentation : d = b  puis b = (b+1)
```

- **" -- " Décrémentation** :

Retranche 1 à l'opérande.

```
Exemples :   a = 14;
             a--;
             a = --a;        // pré-décrémentation : a = (a-1) puis c = a
                             // après exécution c = 12
```

Attention : l'instruction ci-dessous est impossible avec le compilateur CC5X.

```
             d = b--;        // post-décrémentation : d = b  puis b = (b-1)
```

III / Les opérations binaires.

Elles admettent plusieurs opérandes.

- **" + " Addition** :

Effectue la somme algébrique des opérandes.

```
Exemple :    a = b + 3;      // a contient la somme de b et de 3
```

- **" - " Soustraction** :

Effectue la différence des opérandes.

```
Exemple      a = b - 3;   // a contient la différence de b et de 3
```

- **" * " Multiplication** :

Effectue le produit des opérandes.

```
Exemple      a = b * 3;    // a contient 3 fois la valeur de b
```

- **" / " Division** :

Effectue le quotient des opérandes.

```
Exemple      a = b / 3;    // a contient le tiers de b
```

En général, si a est un entier, (3 * a) ne redonnera pas b.

- **" % " Modulo** :

Renvoie le reste de la division entière des opérandes.

```
Exemple      a = 7 % 3;          // a = 7 - ( 3 * 2 ) soit 1
```

- **" && " ET logique** :

Renvoie vrai si les 2 opérandes sont vrais et faux dans les 3 autres cas.

```
Exemple   if  (a && b) c=1;    // c=1 si a et b sont vrais sinon c=0
                               // l'instruction if sera abordée plus loin
```

Attention: sous CC5x, il n'est pas possible d'utiliser le ET logique dans une affectation.

- **"&" ET binaire** :

Effectue un ET logique bit à bit entre les opérandes.

```
Exemple      a = 5;          // a  =  0000 0101
             b = 6;          // b  =  0000 0110
             c = a & b;      // c  =  0000 0100  soit c = 4
```

- **" || " OU logique** :

  Renvoie "1" si 1 ou 2 opérandes sont vrais et "0" dans l'autre cas.
  Exemple if (a || b) c=1; // c = 1 si a ou b sont non nuls , sinon c = 0
  // l'instruction if sera abordée plus loin
  Attention: sous CC5x, il n'est pas possible d'utiliser le OU logique dans une affectation.

- **" | " OU binaire** :

  Effectue un OU logique bit à bit entre les opérandes.

```
Exemple     a = 5;        //     a = 0000 0101
            b = 6;        //     b = 0000 0110
            c = a | b;    //     c = 0000 0111   soit c = 7
```

- **" ^ " OU exclusif binaire** :

  Effectue un OU exclusif bit à bit entre les opérandes.

```
Exemple :   a = 5;        // a =   0000 0101
            b = 6;        // b =   0000 0110
            c = a ^ b;    // c =   0000 0011 soit c = 3
```

- **" << " Décalage de bits à gauche** :

  Décale vers la gauche tous les bits du 1er opérande. Le nombre de décalages est donné par le 2e opérande.

```
Exemple :   a = 11;       // a = 0000 1011 soit a = 0xb
            b = a << 3;   // b = 0101 1000 soit b = 0x58
```

- **" >> " Décalage de bits à droite** :

  Décale vers la droite tous les bits du 1er opérande. Le nombre de décalages est donné par le 2e opérande.

```
Exemple     a = 11    ;   // a = 0000 1011
            b = a >> 3 ;  // b = 0000 0001 soit b = 1
```

IV / Les auto affectations

C'est une écriture condensée liée à une opération binaire et une affectation.
Si l'on écrit a = a + b ; l'écriture de "a" intervient 2 fois, ce qui est inutile.
On pourra écrire : a += b ; // d'abord a+b , ensuite a = (a+b)

```
Exemples    a += 1;       // a = a + 1        Affectation-somme
            b -= 2;       // b = b - 2        Affectation-différence
            c *= 3;       // c = c * 2        Affectation-produit
            d /= 4;       // d = d / 4        Affectation/division
            e %= 5;       // e = e % 5        Affectation-modulo
            f &= 6;       // f = f & 6        Affectation-et
            g |= 7;       // g = g |  7       Affectation-ou
            h ^= 8;       // h = h ^  8       Affectation-ou exclusif
            i <<= 9;      // i = i << 9       Affectation-décalage à gauche
            j >>= 10;     // j = j >> 10      Affectation-décalage à droite
```

V / Les comparaisons

- **" = = " Egalité** :
  Renvoie vrai si les opérandes sont égaux et faux s'ils sont différents.
  Exemple `if( a == 2 ) b=1;` // b vaut 1 si a = 2
  // l'instruction if sera abordée plus loin

- **" ! = " Non égalité** :
  Renvoie vrai si les opérandes sont différents et faux s'ils sont égaux.
  Exemple `if( a != 2 ) b=0;` // b vaut 0 si a différent de 2
  // l'instruction if sera abordée plus loin

- **" > " Plus grand que** :
  Renvoie vrai si le 1er opérande est supérieur au 2e sinon faux.
  Exemple `a = ( 10 > 20 );` // a = 0

- **" < " Plus petit que** :
  Renvoie vrai si le 1er opérande est inférieur au 2e sinon faux.
  Exemple `a = ( 10 < 20 );` // a = 1

- **" >= " Plus grand que ou égal à** :
  Renvoie vrai si le 1er opérande est supérieur ou égal au 2e sinon faux.
  Exemple `if( b >= 2 ) a=1;` // a vaut 1 si b est supérieur ou égal à 2
  // l'instruction if sera abordée plus loin

- **" <= " Plus petit que ou égal à** :
  Renvoie vrai si le 1er opérande est inférieur ou égal au 2e sinon faux.
  Exemple `if( a <= 2 ) b=1;` // b vaut 1 si a est inférieur ou égal à 2
  // l'instruction if sera abordée plus loin

Attention. Il est très **difficile** avec CC5X **d'utiliser une comparaison dans une affectation**.
L'instruction suivante est correcte mais ne fonctionne pas sous CC5X :

```
b =  ( a != 2 );
```

L'instruction suivante fonctionne sous CC5X:

```
a = ( 10 < 20 );        //a=1
```

L'alternative simple et complète (le if)

L'alternative permet de faire un choix de réalisation d'action. Il existe deux types d'alternatives.
Alternative simple : si la condition est vraie, alors faire une action.
Alternative complète : si la condition est vraie, alors faire une action sinon faire une autre action.
A la fin de la réalisation de l'alternative, la suite du programme s'exécute.

En C, l'alternative simple s'écrit comme suit :
If (condition) action;
Dans le cas de plusieurs actions à réaliser :
If (condition) { action1; action2; …}
En C, l'alternative complète s'écrit comme suit :
If (condition) action1; else action2;
Dans le cas de plusieurs actions à réaliser
If (condition) {action1; action2; ...} else {actiona; actionb; ...}

La condition peut être une succession de conditions simples. Si la condition est trop compliqué un message vous avertira.
ex : if (a==3 && b==5) c=2;

L'exemple ci-dessous permettra de clarifier l'utilisation du if.

- Le bouton poussoir 1 allume la led 1 et éteint les autres.
- Le bouton poussoir 2 allume la led 2 et éteint les autres.
- Le bouton poussoir 3 allume toutes les leds.
- Le bouton poussoir 4 éteint toutes les leds.

```
// Attention de respecter les majuscules et minuscules
//-------------------------------E/S-----------------------------------------------
char sortie @ PORTB;
bit inter1 @ RA0;
bit inter2 @ RA1;
bit inter3 @ RA4;
bit inter4 @ RB5;
bit led1 @ RB0;
bit led2 @ RB1;
bit led3 @ RB2;
bit led4 @ RB3;
//-------------------------------Fonction principale-----------------------------
void main(void)
{
        sortie = 0;                     // Initialisation des pattes du microcontroleur
        TRISB = 0b11110000;
        for (;;)  {                     // La suite du programme s'effectue en boucle
                if (inter1==1)          // Si inter1=1 alors allumer led1
                        {
                        led1=1;
                        led2=0;
                        led3=0;
                        led4=0;
                        }
                if (inter2)             // Si inter2=1 alors allumer led2
                        {
                        sortie=0;
                        led2=1;
                        }
```

```
        if (inter3) sortie=0xff;  // Si inter3=1 alors tout allumer
        if (inter4) sortie=0;     // Si inter4=1 alors tout éteindre
                    }
}
```

Explications:

- La vérification de l'état d'un bit peut se faire de deux manières:
  La comparaison : "inter1==1"
  La validation : "inter2" (sous-entendu: si inter2 est vrai)
- Les deux premiers "if" sont écrits de façon différente, la deuxième écriture est préférable car elle occupe moins de place en mémoire.
- Pour mettre toutes les leds à 1, il est nécessaire d'écrire 0xFF sur la sortie. Je rappelle que seuls les bits connectés en sortie sont affectés par cette instruction.

L'alternative multiple (le switch)

Le problème de l'exemple ci-dessus se dessine lorsque l'on appuie sur plusieurs boutons poussoirs à la fois. Il serait aussi peut-être utile de réaliser une action lorsque aucun des if n'est effectué.

Une solution à ces problèmes s'appelle l'alternative multiple

```
switch (variable)  {
        case constante1:
            action1; action2; ...
            break;
        case constante2:
            actiona; actionb; ...
            break;
        default:
            actionx; actiony; ...
            break;
        }
```

Explications:

- Si variable = constante1, action 1 et 2 sont effectuées.
- Le break sert à sortir du switch, donc à ne pas effectuer les autres cas.
  Attention: en l'absence de break toutes les actions du switch s'effectuent à partir du cas actif.
- default indique les actions à effectuer si aucun des cas ne s'est produit.

Remarques :

- Il est possible de faire la même chose que le switch avec des if.

Intérêt du switch sur le if:

- Le principal intérêt du switch est de déterminer un point d'entrée dans une série d'opérations. Il ne faut pas alors utiliser les break.
- Sous CC5X, le code produit avec le switch est plus compact qu'avec les if imbriqués.

L'exemple suivant illustre l'utilisation du switch.

- Le bouton poussoir 1 allume la led 1 et éteint les autres.
- Le bouton poussoir 2 allume la led 2 et éteint les autres.
- Le bouton poussoir 3 allume toutes les leds.
- Le bouton poussoir 4 éteint toutes les leds.
- Dans le cas où aucun ou plusieurs interrupteurs sont actionnés, rien ne se produit.

```
// Attention de respecter les majuscules et minuscules
//------------------------------E/S-----------------------------------------
char sortie @ PORTB;
bit inter1 @ RA0;
bit inter2 @ RA1;
bit inter3 @ RA4;
bit inter4 @ RB5;
bit led1 @ RB0;
bit led2 @ RB1;
bit led3 @ RB2;
bit led4 @ RB3;
//----------------------------Variables generales-------------------------------
char etat_inters;
//----------------------------Fonction principale-------------------------------
void main(void)
{
        sortie = 0;                     // Initialisation des pattes du microcontroleur
        TRISB = 0b11110000;
        etat_inters=0;
        for (;;) {                      // La suite du programme s'effectue en boucle
                etat_inters.0=inter1;   // Mise de l'etat des inter dans la variable etat_inters
                etat_inters.1=inter2;
                etat_inters.2=inter3;
                etat_inters.3=inter4;
                switch (etat_inters){
                        case 1:                 //action sur inter1 uniquement
                                sortie=0;
                                led1=1;
                                break;
                        case 2:                 // action sur inter2 uniquement
                                sortie=0;
                                led2=1;
                                break;
                        case 4:                 // action sur inter3 uniquement
                                sortie=0xff;
                                break;
                        case 8:                 // action sur inter4 uniquement
                                sortie=0;
                                break;
                                }
                }
}
```

Explication:

- dès le début de la boucle infinie, les états des 4 interrupteurs sont mis dans la variable état_inters.
- Si inter1 est actif, etat_inters=1, led1 s'allume.
- Si inter2 est actif, etat_inters=2, led2 s'allume.
- Si inter3 est actif, etat_inters=4, toutes les leds s'allument.
- Si inter4 est actif, etat_inters=8, toutes les leds s'éteignent.
- Si aucun ou plusieurs inters sont actifs, rien n'est modifié, car il n'y a pas d'instruction default.

L'itération 0..N (le while)

L'itération 0..N permet de ne pas faire ou de répéter une ou plusieurs actions en fonction d'une condition.

En C, cette itération s'écrit:

while (condition) action; ou while(condition) {action1; action2; …}

Explication sur le while:

- Tant que la condition est vraie, l'action ou les actions entre crochets sont réalisées.
- Si la condition est fausse, la ou les actions ne sont pas réalisées, la suite du programme s'effectue.
- Si la condition est fausse lors de sa première évaluation, les actions ne seront jamais réalisées.
- Il faut se souvenir que la condition est évaluée avant de commencer la boucle.

Application théorique simple:

```
a=0;
While (a<210)
        {       RA0=1;
                a++;   }
RA0=0;
```

Explications sur l'application:

- Au début, une variable appelé "a" est mise à 0.
- La boucle s'effectue tant que a<210, ce qui est le cas au début puisque a=0.
- La boucle met la patte RA0 à 1 et augmente "a" d'une unité.
- Lorsque a = 210, la boucle ne s'effectue plus, la patte RA0 est mise à nouveau à 0.

Remarques:

- La boucle s'effectue 210 fois, je vous laisse réfléchir pourquoi.
- Nous venons très simplement de réaliser une temporisation qui met à 1 la patte RA0 pendant un certain temps.

Quelques idées d'utilisation:

- Le while peut servir d'autorisation, d'interdiction de fonctionnements en boucle.
- Il est pratique pour réaliser des actions tout en attendant un événement.
- Il peut servir aussi à attendre qu'une entrée ait un état déterminé pour effectuer la suite du programme. Il faut a ce moment ne pas mettre d'action.

```
action1;
while(!RB0) ;                   //attente que RB0 passe à 1
action2;
```

action1 se réalise. Pour que action2 se réalise, il faut attendre que RB0 passe à 1.

L'exemple suivant va nous permettre de comprendre l'utilité de l'itération 0..N.

- Le bouton poussoir 1 allume les leds 1 et 2 et éteint les deux autres.
- Le bouton poussoir 2 allume les leds 3 et 4 et éteint les deux autres.
- Le bouton poussoir 3 éteint toutes les leds.
- Dans le cas où aucun ou plusieurs interrupteurs sont actionnés, rien ne se produit.
- Le bouton poussoir 4 permet le fonctionnement ci-dessus lorsqu'il est actionné.

```
// Attention de respecter les majuscules et minuscules
//------------------------------E/S-----------------------------------------
char sortie @ PORTB;
bit inter1 @ RA0;
bit inter2 @ RA1;
bit inter3 @ RA4;
bit inter4 @ RB5;
bit led1 @ RB0;
bit led2 @ RB1;
bit led3 @ RB2;
bit led4 @ RB3;
//----------------------------Variables generales-------------------------------
char etat_inters;
//-----------------------------Fonction principale-----------------------------
void main(void)
{
        sortie = 0;                     // Initialisation des pattes du microcontroleur
        TRISB = 0b11110000;
        etat_inters=0;
        For(;;)
        {
        while (inter4)                  // La suite du programme s'effectue si inter4 est actionne
                {
                etat_inters.0=inter1;   // Mise de l'etat des inter dans la variable etat_inters
                etat_inters.1=inter2;
                etat_inters.2=inter3;
                switch (etat_inters)
                        {
                        case 1:                 //action sur inter1 uniquement
                                sortie=0;
                                led1=1;
                                led2=1;
                                break;
                        case 2:                 // action sur inter2 uniquement
                                sortie=0;
                                led3=1;
                                led4=1;
                                break;
                        case 4:                 // action sur inter3 uniquement
                                sortie=0;
                                break;
                        }
                }
        }

}
```

Explications:

- Les fonctions réalisées par les boutons poussoirs 1, 2 et 3 ne peuvent se faire que lorsque inter4 est actionné.
- Nous utiliserons une itération 0..N dans laquelle nous placerons les fonctions des 3 premiers boutons poussoirs. La condition de réalisation de l'itération sera le bouton poussoir 4 activé.

L'itération 1..N (le do while)

L'itération 1..N permet de répéter au moins une fois une ou plusieurs actions en fonction d'une condition.

En C, cette itération s'écrit:

do action; while (condition); ou do {action1; action2; …} while (condition);

Explications:

- L'action ou les actions entre crochets sont réalisées tant que la condition est vraie.
- L'évaluation de la condition se fait après réalisation de ou des actions.

Application théorique simple:

```
a=0;
do
        {       RA0=1;
                a++;   }
While (a<210);
RA0=0;
```

Explications sur l'application:

- Au début, une variable appelé "a" est mise à 0.
- Puis, la patte RA0 est mise à 1, et a est augmenté d'une unité.
- a est alors comparé à 210. Si a<210, les actions entre crochets s'effectuent à nouveau.
- Au moment où a=210, les actions entre crochets ne s'effectuent plus, la suite du programme peut se réaliser.
- La patte RA0 est alors mise à nouveau à 0.

Remarque:

- La boucle s'effectue 211 fois, je vous laisse réfléchir pourquoi.
- Nous venons très simplement de réaliser une temporisation qui met à 1 la patte RA0 pendant un certain temps.

Exemple:

- Le microcontrôleur observe si les boutons poussoirs 1, 2 et 3 sont actionnés.
- Lorsque l'on active le bouton poussoir 4, l'état des leds est mis à jour, comme suit.
- La led 1 s'allume si le bouton poussoir 1 a été actionné.
- La led 2 s'allume si le bouton poussoir 2 a été actionné.
- La led 3 s'allume si le bouton poussoir 3 a été actionné.
- Les boutons poussoirs ne seront observés à nouveau que lorsque inter4 est relâché.

```
// Attention de respecter les majuscules et minuscules
//------------------------------E/S-------------------------------------------
char sortie @ PORTB;
bit inter1 @ RA0;
bit inter2 @ RA1;
bit inter3 @ RA4;
bit inter4 @ RB5;
bit led1 @ RB0;
bit led2 @ RB1;
bit led3 @ RB2;
bit led4 @ RB3;
//----------------------------Variables generales-------------------------------
char etat_inters;
//----------------------------Fonction principale-------------------------------
void main(void)
{
        sortie = 0;                     // Initialisation des pattes du microcontroleur
        TRISB = 0b11110000;
        etat_inters=0;
        for (;;)
        {
        do                      // boucle de memorisation d'actions sur les boutons poussoirs 1 2 et 3
                {
                if (inter1) etat_inters.0=1;
                if (inter2) etat_inters.1=1;
                if (inter3) etat_inters.2=1;
                }
        while (!inter4);
        led1=etat_inters.0;             // mise a jour des leds
        led2=etat_inters.1;
        led3=etat_inters.2;
        etat_inters=0;
        while (inter4) ;                // attente que l'on relache inter4
        }
}
```

Explications:

- Les actions sur les boutons poussoirs 1, 2 et 3 sont mémorisées tant que l'on n'actionne pas inter4.
- Inter4 pas actionné s'écrit: !inter4
- Une action sur inter4 stoppe la boucle et met à jour les leds.
- Pour attendre que l'on relâche inter4, il suffit de faire une boucle while sans action.

L'itération avec variable de contrôle (le for)

Le for permet de faire une boucle contrôlée grâce à une variable et des conditions: début de boucle, condition de réalisation de boucle, incrémentation.

En C, cette itération s'écrit:

For(début; comparaison; incrémentation) action;
For(début; comparaison; incrémentation) { action1; action2; …}

avec

- "début" : instruction d'affectation selon une écriture valide.
- "comparaison" : instruction de comparaison selon une écriture valide.
- "incrémentation" : incrémentation ou décrémentation de variable.

Explications:

- L'action ou les actions entre crochets se répètent de l'affectation "début" tant que "comparaison" est vraie. A chaque boucle "incrémentation" se réalise.

Application théorique simple:

```
Char a;
for (a=3; a<10; a++}
        {       RA0=!RA0;
                RA1=!RA1}
```

Explications sur l'application:

- Une variable "a " est créée pour réaliser une boucle for.
- En entrant dans la boucle, la variable "a" est mise à 3.
- Puis la condition a<10 est évaluée. Celle-ci est vraie puisque a=3, les actions entre crochets peuvent se réaliser.
- Ces actions inversent le niveau logique des pattes RA0 et RA1.
- Lors de chaque fin de réalisation des actions de la boucle, "a" est incrémenté d'une unité.
- La condition a<10 est à nouveau évaluée et le cycle recommence.
- A force d'incrémenter a, la comparaison a<10 deviendra fausse, la boucle sera alors terminée, la suite du programme pourra s'exécuter.

Remarques:

- La boucle s'effectue 7 fois, je vous laisse réfléchir pourquoi.
- L'écriture en C est très souple. Il est alors possible d'omettre une ou plusieurs conditions.

```
for ( ; a<10; a++)  // "a" est affecté avant l'entrée dans la boucle.
for (a=3; a<10 ; )  // "a" doit être mis à jour dans la boucle.
for ( ; a<10 ; )  // le for ne maîtrise que la condition de boucle.
for ( ; ; ;)  // aucune condition, donc boucle infinie.
```

- La boucle suivante ne fait aucune action, elle permet de perdre du temps

```
for ( a=0; a<250; a++) ;
```

Exemple:

- Le microcontrôleur observe si les boutons poussoirs 1, 2 et 3 et 4 sont actionnés.
- Lorsque 3 boutons poussoirs ont étés actionnés (même plusieurs fois) l'affichage se met à jour comme suit:.

La led 1 s'allume, si le bouton poussoir 1 a été actionné.
La led 2 s'allume, si le bouton poussoir 2 a été actionné.
La led 3 s'allume, si le bouton poussoir 3 a été actionné.
La led 4 s'allume, si le bouton poussoir 4 a été actionné.

```
/char etat_inters;
char action;
//------------------------------Fonction principale-----------------------------
void main(void)
{
        sortie = 0;                     // Initialisation des pattes du microcontroleur
        TRISB = 0b11110000;
        etat_inters=0;
        for (;;)
        {
        for (action=1; action<4; )      // boucle de memorisation d'actions sur les boutons poussoirs
           {
                if (inter1&&etat_inters.0==0) {etat_inters.0=1; action++;}
                if (inter2&&etat_inters.1==0) {etat_inters.1=1; action++;}
                if (inter3&&etat_inters.2==0) {etat_inters.2=1; action++;}
                if (inter4&&etat_inters.3==0) {etat_inters.3=1; action++;}
                }
        led1=etat_inters.0;             // mise a jour des leds
        led2=etat_inters.1;
        led3=etat_inters.2;
        led4=etat_inters.3;
        etat_inters=0;
        }
}
```

Explications:

- Dans la boucle for, la première action sur chaque bouton poussoir est mémorisée dans état_inters et une variable "action" est incrémentée.
- lorsque 3 boutons poussoirs ont étés actionnés, on sort de la boucle, les leds sont mises à jour.

Pour quelques neurones de plus

Nous venons de voir les principales structures du langage C. Il en existe encore beaucoup.
Pour développer un programme en C destiné à un microprocesseur, ce que nous venons d'apprendre suffit largement.

Si vous voulez approfondir le sujet, je vous conseille de travailler les structures suivantes grâce au manuel d'utilisation de CC5X :

break;
continue;
return;
goto.

Ces structures permettent de rendre encore plus souple l'utilisation des alternatives et des itérations. Ces structures peuvent être d'un grand secours dans des programmes complexes, pour **gérer** les erreurs, les cas non pris en compte par le programme. on appelle souvent ces problèmes **des exceptions**.

Conclusion

Avec ce troisième fascicule vous êtes enfin autonome pour écrire vos premiers programmes. Le niveau de connaissances que vous possédez est largement suffisant pour la plupart des applications où la gestion du temps n'a pas d'importance. Maintenant, seule l'expérience fera la différence. Prenez le temps de lire des programmes, de vous casser les dents sur vos propres réalisations.

Avec le quatrième didacticiel, vous serez capable de gérer le temps. La gestion du temps est théoriquement assez facile à comprendre, mais dans la pratique plutôt complexe à cause de l'éventail des possibilités offertes par les microcontrôleurs dans ce domaine.
Je conseille donc de passer à la suite de ces cours que si vous possédez parfaitement les différentes notions vues jusqu'ici.