INRACI	Informatique Appliqué **Syllabus d'exercices**	5ème EE 2010-2011

Table des matières

Mon premier programme (sans boucle infinie)

1er programme (avec ici comme noms de fichiers « prog.h » et « prog.c ») :

« prog.h », dans le cas d'un PIC18F2620

```
__CONFIG(1, HS&IESODIS&FCMDIS);
__CONFIG(2, BORDIS&PWRTEN&WDTDIS );
__CONFIG(3, PBANDIS);                          //enlever cette ligne si on a un PIC18F2520
__CONFIG(4, LVPDIS&XINSTDIS&STVRDIS&LPT1DIS);
__CONFIG(5, UNPROTECT);
__CONFIG(6, WRTEN);                            //enlever cette ligne si on a un PIC18F2520

extern void init(void);
```

« prog.c »

```
#include<pic18.h>
#include"prog.h"

#define  BP1 RA0
#define  BP2 RA1
#define  BP3 RA2
#define  BP4 RA3

#define  LED1 RB7
#define  LED2 RB6
#define  LED3 RB5
#define  LED4 RB4
#define  LED5 RB3
#define  LED6 RB2
#define  LED7 RB1
#define  LED8 RB0

#define  LEDS PORTB

void main(void)
{
init();
LEDS=0b11111111;
while(1) ;                    // dans un 1er temps on utilise une ligne dont la fonction est « stop ici »
}//fin du main

/*************************************************************/
void init()
        {
        TRISA=0b11101111;
        ADCON1=0x0f;
        PORTA=0;
        TRISB=0b00000000;
        PORTB=0;
        TRISC=0b00000000;
        PORTC=0;
        }
```

Résumé de la syntaxe en LDA et en C

Syntaxe de base pour la déclaration la lecture, l'affectation et l'écriture :

Exemple de séquence en LDA :	Exemple de séquence en C :
Réels montant_place, taux	float montant_place, taux ;
Réels interet, valeur_acquise	float interet, valeur_acquise ;
Lire montant_place, taux	// je lis montant_place, taux
interet ← montant_place*taux/100	interet = montant_place*taux/100 ;
valeur_acquise ← montant_place + interet	valeur_acquise = montant_place + interet ;
Ecrire interet, valeur_acquise	// j'écris interet, valeur_acquise

Syntaxe des types les plus utilisés (PICC18) :

LDA	Syntaxe en C	Description	Nombre d'octets
Entier	Long	entier long signé	4 octets
Entier	Unsigned long	entier long non signé	4 octets
Entier	Int	entier standard signé	2 octets
Entier	Unsigned int	entier positif	2 octets
Entier	Char	caractère signé	1 octet
Entier	Unsigned char	caractère non signé	1 octet
Bit	Bit	booléen	1/8 d'octet
Réel	Float	réel standard	3 octets

Expression d'une constante en différentes bases en C :
PORTB=255; ou PORTB=0b11111111; ou PORTB=0xFF;

Opérateurs arithmétiques en C (hiérarchie habituelle) :
Sur des réels : + - * /
Sur des entiers : + - * / (quotient de la division) et % (reste de la division)

Opérateurs logiques en C :
ET → && OU → || NON→ !

incrémentation et décrémentation en C :
i++ ; //ou i=i+1 ; i-- ; //ou i=i-1 ;

Opérateurs manipulateurs de bits en C :

Et	Ou	Xor	Complément à 1	Décalage gauche	Décalage droit
&	|	^	~	<<	>>

Opérateurs de comparaison en C :
Coïncidence : « == », différent de : « != », autres : <, <=, >, >=

Syntaxe directives #define et #include :

#define BP1 RA0	#define PI 3.14	#include <pic18.h>	#include "delay.h"

Structure de choix en LDA

si *expression* alors
    //bloc d'action 1
sinon
    //bloc d'action 2
fsi

Structure de choix en C

```
if(expression)
    {//bloc d'action 1
    }
else
    {//bloc d'action 2
    }
```

Choix multiple en LDA

selon que
    i=1 faire
        //séquence 1 d'instructions
oq *i=2* faire
        //séquence 2 d'instructions
oq *i=3* faire
        //séquence 3 d'instructions
autrement
    //séquence 4 d'instructions
fselon

Choix multiple en C

```
switch (i)
    {case 1 : //séquence 1 d'instructions
              break;
     case 2 : //séquence 2 d'instructions
              break;
     case 3 : //séquence 3 d'instructions
              break;
     default :
          //séquence 4 d'instructions
    }
```

« tant que » en LDA

tant que *expression* faire
    //bloc d'actions
ftant

« tant que » en C

```
while (expression)
    {//bloc d'actions
    }
```

«répéter… tant que » en LDA

répéter
    //bloc d'actions
tant que *expression*

«répéter…tant que » en C

```
do  {//bloc d'actions
    }
while (expression) ;
```

«boucle pour » en LDA

pour *i* de 1 à 10 faire
    //bloc d'actions
fpour

«boucle pour » en C

```
for (i=1 ; i<=10 ; i++)
    {//bloc d'actions
    }
```

Exemples de déclarations de fonctions :

float moyenne (int, int);	`void lcd_putch(char);`	`void lcd_clear(void);`

Syntaxe pour l'implémentation de fonctions :

float moyenne (int a, int b)
{
float moy ;
moy=(a+b)/2 ;
return moy;
}
	`void lcd_putch(char c)`
	{
	// utilisation de « c »
	// « c » pas modifiable
	//pas de return
	}
		`void lcd_clear(void);`
		{
		//lignes d'instructions
		//pas de return
		}

Utilisation du sprintf(…) :

```
#include        <stdio.h>
…
near char Ligne1[16];
…
sprintf(Ligne1,"  %2d bla %2d  bla bla ", var1,var2);
lcd_goto(0);              // si on suppose var1=5 et var2=8, sera alors
lcd_puts(Ligne1);       // affiché :  «  5 bla  8 blabla  »
```

Utilisation de la librairie delay

```
#include  "delay.h"
…
int t=300 ;
delay_ms(t);
delay_ms(200);
delay_us(100);
```

Utilisation de la librairie lcd

```
#include  "lcd.h"
…
lcd_init() ;
lcd_clear() ;
lcd_goto(0x40);
lcd_puts(“coucou”);
```

Structure de base d’un programme :

```
#include        <pic18.h>
#include        <stdio.h>
#include        "lcd.h"
#include        "delay.h"

#define  BP1 RA0
#define  BP2 RA1
#define  BP3 RA2
#define  BP4 RA3

near char Ligne1[16];
near char Ligne2[16];
void main(void)
{
init() ;
while(1)
     {
     }
}
```

Implémentation de la fonction init() :

```
    void init()
    {   TRISA=0b11101111;
        ADCON1=0x0f;
        PORTA=0;
        TRISB=0b00000000;
        PORTB=0;
        TRISC=0b00000000;
        PORTC=0;
        lcd_init();
        lcd_clear();
    }
```

Enoncés d'exercices

Série 1 : L'informatique embarquée

Exposez les schémas bloc des dispositifs suivants :

1.1) Une calculette
1.2) Un multimètre
1.3) Un radio-réveil
1.4) Un robot suiveur de ligne
1.5) Une imprimante
1.6) Une petite station météo
1.7) Une alarme (sonore et visuelle)

Série 2 : La base de la programmation

Exercices liés à la notion de variable et son type

2.1) Expliquez brièvement ce qu'est une variable

2.2) Quel nom et catégorie de type choisissez-vous pour une variable destinée à :
- 2.2.1) Contenir une information pour un robot qui participe à un concours de robotique. Le robot peut jouer le côté bleu ou le côté rouge, c'est tiré au hasard. Une fois la configuration fixée, l'information de couleur doit être disponible au sein du programme à l'aide d'une variable.
- 2.2.2) L'affichage des secondes pour un chronomètre.
- 2.2.3) Contenir la valeur du nombre π
- 2.2.4) Contenir le résultat de la somme de deux entiers
- 2.2.5) Contenir le résultat de la moyenne de deux entiers
- 2.2.6) L'affichage de la pression atmosphérique en Pascal à l'unité près
- 2.2.7) Indiquer si on est au sein du menu principal ou non
- 2.2.8) Servir de compteur interne pour répéter 10 fois une action
- 2.2.9) Choisir le mode de configuration allumé/éteint de l'écran
- 2.2.10) La température dans une pièce

2.3.) À l'aide du nom de la variable essayer de décrire son rôle, le nom de la variable est il bien choisi ?
- 2.3.1) nb_ms
- 2.3.2) produit
- 2.3.3) u
- 2.3.4) nb_impulsions
- 2.3.5) nb_enfoncements
- 2.3.6) nombre_de_fois_que_le_bouton1_a_ete_enfonce
- 2.3.7) code_est_valide
- 2.3.8) m

2.3.9) vitesse
2.3.10) numero_strategie

2.4) A l'aide du nom de la variable essayer de deviner sa famille de type, le nom de la variable est il bien choisi ?
2.4.1) mode_auto
2.4.2) numero_menu
2.4.3) temp
2.4.4) x
2.4.5) vitesse
2.4.6) consigne_moteur
2.4.7) i
2.4.8) type_trajectoire
2.4.9) sens_deplacement
2.4.10) puissance_consommee

Exercices liés à l'affectation en LDA et en C et les opérateurs arithmétiques

2.5) Expliquez ce qu'est une affectation.

2.6) Ecrivez en LDA et en C les lignes correspondantes :
2.6.1) Affecter la valeur 0 à la variable *nb_enfoncements*
2.6.2) Mettre 10 dans *numero_menu*
2.6.3) Mettre *x* dans *y*
2.6.4) Affecter le contenu de la variable *x* à la variable *y*
2.6.5) Affecter à la variable *result* la somme de *nb1* et de *nb2*
2.6.6) Affecter à la variable *moyenne* la moyenne des variables *nb1* et *nb2*
2.6.7) Mettre dans la variable *sens_deplacement* une valeur qui correspond à avancer dans la mesure où il a été convenu que la valeur '0' correspond à reculer et la valeur '1' correspond à avancer.
2.6.8) Affecter à la variable *i*, son ancienne valeur augmentée de 1
2.6.9) Incrémenter la variable *sec*
2.6.10) Décrémenter la variable *nb_sec_avant_fin_match*

2.7) Commentez (expliquez en français) les lignes de C suivantes :
2.7.1) i = i +1 ;
2.7.2) result = (nb1 + nb2)/2 ;
2.7.3) mode_auto = 1 ;
2.7.4) nb_enfoncements= nb_enfoncements+1 ;
2.7.5) cpt = cpt -10 ;

2.8) Commentez (expliquez en français) les lignes de LDA suivantes :
2.8.1) nb_BP1 ← (nb_BP1 + 1)
2.8.2) code_est_valide ← 0
2.8.3) sens_deplacement ← 1
2.8.4) LED1 ← 0
2.8.5) j ← j-1

2.9) Quel sera le contenu de la variable *result* à la fin de la dernière instruction ?

2.9.1)	2.9.2)	2.9.3)	2.9.4)
result = 10 ; i = 20 ; i = i -1 ; relsult = result +i ;	i = 5 ; result = 2*i + 1 ; i = i+10 ; result = 3*result – i ;	j= 50 ; i = 10 ; i= i +j -10 ; result = 2*i - j ;	n1=12 ; n2=9 +n1 ; result = (n1+n2)/2 ;

2.10.) Ecrivez en LDA et en C les lignes correspondantes :

2.10.1) Affecter le produit de la variable *nb* et de la constante 10 à la variable *result*

2.10.2) Ajouter 10 à la variable *i*

2.10.3) Mettez dans la variable *code_valide* une valeur qui indique que le code est ok

2.10.4) On allume la LED3

2.10.5) On affecte à la variable *nb_enfoncements* son ancienne valeur ajoutée de 1

Série 3 : La séquence

Exercices sur la séquence en ordinogramme

3.1) Faites l'ordinogramme complet qui permet de calculer la somme de trois nombres entiers

3.2) Faites l'ordinogramme complet qui permet de calculer le carré d'un nombre entier

3.3) Sachant que pour les ordinogrammes, pour attendre un délai d'une seconde, il suffit d'écrire : ATTENDRE 1s. au sein d'un rectangle.
Pour allumer une led, il suffit d'affecter la valeur 1 à la led qu'il faut allumer.
Sachant cela, faites l'ordinogramme complet qui gère 2 clignotements consécutifs de la LED1.

3.4) Sachant que pour les ordinogrammes, pour afficher du texte, il suffit d'écrire : AFFICHER « … » au sein d'un rectangle, avec le texte entre « et »
Sachant cela, faites l'ordinogramme complet qui gère 3 alternances d'affichage « coucou » … « c'est moi »

3.5) En supposant que l'on ne puisse pas agir sur l'ensemble des LEDS en même temps (mais qu'on agisse seulement individuellement sur chaque LED) et qu'on ne dispose pas de structure informatique (if, while, for,…), faites l'ordinogramme complet de la gestion d'un trajet chenillard.
Un chenillard est simplement un jeu de lumière qui donne l'impression qu'une LED se déplace de gauche à droite. C'est d'abord la LED1 qui est allumée (les autres sont éteintes), puis c'est la LED2 qui est allumée (les autres sont éteintes),…

3.6) Faites l'ordinogramme complet qui permet de calculer la moyenne de deux nombres entiers mais ajoutez des affichages de textes destinés à l'utilisateur.

3.7) Faites l'ordinogramme complet qui permet de calculer le carré d'un nombre entier mais ajoutez des affichages de textes destinés à l'utilisateur.

3.8) Faites l'ordinogramme complet sachant que :
Une personne (l'utilisateur) voudrait s'acheter une nouvelle voiture. On sait que le vendeur est prêt à accepter que le paiement de la voiture se fasse sur un nombre d'années allant de 1 à 8 et ce sans intérêts supplémentaires. L'acheteur voudrait utiliser un programme informatique qui lui indique combien il lui faudra payer chaque mois. Bien sûr l'utilisateur doit introduire le montant de la voiture et le nombre d'années pendant lesquelles le paiement aura lieu.

3.9) Faites l'ordinogramme complet sachant que :
Un professeur (l'utilisateur) voudrait utiliser un programme informatique qui puisse calculer le total de la période d'un élève. On sait que chaque période contient deux interros et que chaque interro est cotée sur un certain nombre de points. L'utilisateur doit alors entrer les valeurs des deux interros et les cotes maximales associées.
Le programme doit fournir le résultat de la période (/60).

Exercices sur la séquence en C (sur papier)

3.10) Sachant qu'à ce stade nous ne sommes pas encore capable d'effectuer une lecture ou une écriture dans le cadre de l'utilisation de la carte info, nous nous contenterons de mettre un commentaire pour préciser qu'il s'agit d'une lecture ou une écriture. Dans ces conditions développez un programme en C qui permet de calculer la somme de trois nombres entiers.

3.11) Sachant qu'à ce stade nous ne sommes pas encore capable d'effectuer une lecture ou une écriture dans le cadre de l'utilisation de la carte info, nous nous contenterons de mettre un commentaire pour préciser qu'il s'agit d'une lecture ou une écriture. Dans ces conditions, développez un programme en C qui permet de calculer le carré d'un nombre entier.

Exercices sur la séquence en C (sur PC)

Comme MPLAB redémarre le programme lorsqu'il arrive à la dernière instruction, dans un 1er temps nous écrirons à la dernière ligne « while(1) ; » ce qui signifie « stop ici ». Nous verrons plus tard qu'il ne faut pas inclure cette ligne mais utiliser une boucle infinie « while(1) {}».

Sachant qu'avec les librairies que nous utilisons et l'utilisation de la carte info :
L'appel de la fonction init() ; doit figurer en début de programme
Pour attendre une seconde il suffit d'écrire delay_ms(1000) ;
Pour allumer la LED1 il suffit d'écrire LED1=1 ;
Pour éteindre la LED1 il suffit d'écrire LED1=0 ;
Pour allumer les LEDS 2, 3 et 4 d'un coup et éteindre les autres on écrit :
LEDS=0b01110000 ;

3.12) ☺ Toutes leds
Développer un programme qui s'initialise puis qui allume toutes les leds.
La solution de cet exercice est donnée à la page 2.

3.13) ☺ Led sur deux

Développer un programme qui s'initialise puis qui allume une led sur deux.

3.14) ☺ Quatre premières leds

Développer un programme qui s'initialise puis qui allume les 4 1ères leds et éteint les 4 autres.

3.15) ☺ Allume 2 leds milieu

Développer un programme qui s'initialise puis qui allume les leds qui correspondent à RB3 et RB4 (toutes les autres leds doivent être éteintes).

3.16) ☺ 2 cligno LED1

Développez un programme en C qui gère 2 clignotements consécutifs de la LED1.

Sachant qu'avec les librairies que nous utilisons et l'utilisation de la carte info :

Pour afficher du texte sur la 1ère ligne de l'écran il suffit d'écrire :

```
lcd_goto(0x00) ;
lcd_puts(″…″) ;                 //où le texte est situé entre ″ et ″
```

Pour effacer la 1ère ligne il suffit d'écrire :

```
lcd_goto(0x00) ;
lcd_puts(″                ″) ;   //où il y a 16 espaces entre ″ et ″
```

Pour afficher du texte sur la 2ème ligne de l'écran il suffit d'écrire :

```
lcd_goto(0x40) ;
lcd_puts(″…″) ;                 //où le texte est situé entre ″ et ″
```

Pour effacer la 2ème ligne il suffit d'écrire :

```
lcd_goto(0x40) ;
lcd_puts(″                ″) ;   //où il y a 16 espaces entre ″ et ″
```

3.17) ☺ 3 alternances coucou - c'est moi

Développer un programme en C qui gère 3 alternances d'affichage « coucou » ... « c'est moi »

3.18) ☺ chenillard droite à gauche

Développer un programme en C qui gère un trajet chenillard de droite à gauche.

3.19) ☺ ☺ moyenne sans lecture ni écriture

Développez un programme en C qui permet de calculer la moyenne de deux nombres entiers.
On se contentera de remplacer les instructions de lecture et d'écriture par des commentaires.
Votre programme ne pourra hélas pas être testé car nous ne sommes pas encore capables de gérer les instructions de lecture et l'écriture appliquées à la carte info. La compilation peut néanmoins être faite et la structure du programme peut être vérifiée par vos professeurs.

3.20) ☺ ☺ carré sans lecture ni écriture

Développez un programme en C qui permet de calculer le carré d'un nombre entier.
On se contentera de remplacer les instructions de lecture et d'écriture par des commentaires.
Votre programme ne pourra hélas pas être testé car nous ne sommes pas encore capables de gérer les instructions de lecture et l'écriture appliquées à la carte info. La compilation peut néanmoins être faite et la structure du programme peut être vérifiée par vos professeurs.

Série 4 : La structure de choix simple

4.1) Montrez le morceau d'ordinogramme ainsi que les lignes en C qui correspondent aux explications données ci-dessous :

4.1.1) On commence par lire le contenu de la variable *nb*. Ensuite, si *nb* est strictement positif, on incrémente la variable *i* ; dans les autres cas on ne fait rien.

4.1.2) On teste le contenu de BP1 et on affiche son état d'enfoncement

4.1.3) Après une lecture des variables *a* et *b*, on affecte à la variable *min* le minimum des deux variables.

4.1.4) Après une lecture des variables *a,b* et c, on affecte à la variable *max* le maximum des trois variables.

4.2) Faites l'ordinogramme complet ainsi que le programme en C qui répond à l'énoncé ci-dessous (en C nous nous contenterons de commentaires pour lire et écrire) :

Une personne compte aller chaque jour de la semaine au magasin. A la fin de la semaine la personne possède donc 5 tickets de caisse. L'idée est alors que la personne utilise un programme informatique pour lui faire le calcul du prix total qui aura effectivement été payé.
Il faut cependant prendre en compte une ristourne éventuelle :
En effet le vendeur s'est engagé à faire une ristourne de 20% du montant total dans la mesure où celui-ci dépasse 500 euros.

4.3) On essaye ici d'analyser ce qu'il se passe, il ne s'agit pas d'exemples pratiques. Quel sera le contenu de la variable *result* à la fin de la dernière instruction ?

4.3.1)	4.3.2)	4.3.3)	4.3.4)

result = 20 ;
//je lis i
j= i +1 ;
if(i >=j)
{
result =10 ;
}
	i=10 ;
	j= 2*i -10 ;
	if(i != j)
	{
	result =8 ;
	}
	else
	{
	result =50 ;
	}
		//je lis i
		j= 10 ;
		if(i == i)
		{
		result =j ;
		}
		else
		{
		result =5 ;
		}
			i=0 ;
			j= i+1 ;
			if((i+3) >= (j+2))
			{
			result =j+10;
			}
			else
			{
			result =i +5 ;
			}

4.4) En reprenant les quatre exemples de l'exercice C, essayez de donner les morceaux d'ordinogramme équivalents.

4.5) Sans utiliser d'opérateurs de comparaison, développez le programme en C et l'ordinogramme qui permet d'allumer la LED1 que si BP1 et BP2 sont tous les deux enfoncés. Il faut utiliser des si-imbriqués.

4.6) Sans utiliser d'opérateurs de comparaison, développez le programme en C et l'ordinogramme qui permet d'allumer la LED1 que si BP1 ou BP2 est enfoncé. Il s'agit d'un OU non exclusif. Il faut utiliser des si-imbriqués.

4.7) Sans utiliser d'opérateurs de comparaison, développez le programme en C et l'ordinogramme qui permet d'allumer la LED1 que si seulement BP1 ou seulement BP2 est enfoncé. Il s'agit d'un OU exclusif. Il faut utiliser des si-imbriqués.

Exercices sur PC

Comme MPLAB redémarre le programme lorsqu'il arrive à la dernière instruction, nous continuerons à écrire à la dernière ligne « while(1) ; » ce qui signifie « stop ici ». Nous verrons plus tard qu'il ne faut pas inclure cette ligne mais utiliser une boucle infinie « while(1) {}».
Dans ces conditions, il faut garder à l'esprit que lors de l'exécution du programme chaque condition examinée au sein d'une structure de choix se fera à un moment bien précis. Ce qui veut dire par exemple qu'un bouton-poussoir enfoncé ne pourra être détecté que si le bouton était déjà enfoncé avant d'examiner la condition.
On testera alors le programme en appuyant sur le(s) bouton(s) juste avant de provoquer un reset du programme.

4.8) ☺ ☺ Toutes leds si BP

Développer un programme qui s'initialise puis qui éteint toutes les leds s'il n'y a pas de bouton-poussoir enfoncé mais qui allume toutes les leds dès qu'au moins un bouton-poussoir est enfoncé.

4.9) ☺ ☺ 4 1ères leds reflètent BP

Développer un programme qui s'initialise puis qui éteint toutes les leds s'il n'y a pas de bouton-poussoir enfoncé. Par contre dès qu'il y a un ou plusieurs boutons-poussoirs qui sont enfoncés, il faut que la ou les leds correspondantes s'allument. Donc si on appuie sur les boutons-poussoirs 1 et 3, il faut que les leds en position 1 et 3 s'allument.

4.10) ☺ ☺ X 1ères leds (X nb BP enfoncés)

Développer un programme qui s'initialise puis qui allume les x 1ères leds, x étant le nombre de boutons-poussoirs enfoncés. Il faut donc que si deux boutons-poussoirs sont enfoncés (peu importe lesquels), les deux 1ères leds s'allument.

4.11) ☺ ☺ Nb BP enfoncés en binaire

Développer un programme qui s'initialise puis qui indique en binaire (à l'aide des leds) le nombre de boutons-poussoirs enfoncés. Il faut donc que si quatre boutons-poussoirs sont enfoncés, les leds indiquent (00000100).

4.12) ☺ ☺ X dernières leds (X nb BP enfoncés)

Développer un programme qui s'initialise puis qui allume les x dernières leds, x étant le nombre de boutons-poussoirs enfoncés. Il faut donc que si deux boutons-poussoirs sont enfoncés (peu importe lesquels), les 2 dernières leds s'allument.

4.13) ☺ ☺ 8 leds reflètent BP

Développer un programme qui s'initialise puis qui allume les leds de manière à refléter les états d'enfoncement des boutons-poussoirs, on va dire que l'état de chaque bouton-poussoir est reflété par un groupe de 2 leds. BP1 influence les 2 1ères leds, BP2 les 2 suivantes…
Il faut alors que si on enfonce par exemple BP1 et BP3, se sont les leds qui correspondent à RB7, RB6, RB3 et RB2 qui s'allument (toutes les autres leds doivent être éteintes).

4.14) ☺ Trimmer dernière led

Développer un programme qui consiste à allumer la dernière led si la résistance variable est réglée tout à fait à gauche et à éteindre le dernière led si la résistance variable est réglée tout à fait à droite. Toutes les autres leds sont toujours éteintes.

4.15) ☺ ☺ Trimmer et X leds (X nb BP enfoncés)

Développer un programme qui consiste à allumer soit les X 1ères leds soit les X dernières leds suivant que le trimmer soit respectivement tout à fait à gauche ou tout à fait à droite.
X correspond au nombre de boutons-poussoirs enfoncés (parmi BP2, BP3 et BP4), les leds pas concernées doivent rester éteintes.

4.16) ☺ ☺ ☺ Trimmer, X leds et BP4 (X nb BP enfoncés)

Lorsque BP4 est relâché, le programme doit se comporter de la même manière que l'exercice 4.15 avec X qui est cette fois le nombre de boutons-poussoirs enfoncés parmi BP2 et BP3 uniquement. Si BP4 est enfoncé on a la même situation mais où chaque led change d'état.

Série 5 : La « tant que »

Pour les exercices ci-dessous, continuez à mettre l'instruction « while(1) ; » comme dernière instruction. Nous verrons un peu plus loin que cette instruction (qui signifie « stop ici ») ne devrait pas figurer dans un programme classique qui devrait plutôt contenir une boucle infinie de type while(1){}. On place actuellement l'instruction « while(1) ; » à la dernière instruction afin d'éviter que MPLAB redémarre le programme une fois arrivé à la fin ; on se rapproche ainsi d'une représentation sous forme d'ordinogramme telle que vue jusqu'à présent (qui se termine par l'entité « fin »). Plus tard on utilisera une boucle infinie de type « while(1) {} » (sans le point-virgule) de manière à ne jamais sortir du programme principal.

5.1) Exposez l'ordinogramme qui permet de faire clignoter la LED1 tant que BP1 reste enfoncé.

5.2) Développez en C le programme qui correspond à l'ordinogramme exposé en 5.1)

5.3) Exposez l'ordinogramme qui permet de gérer l'alternance d'affichage :
« BP1 enfoncé », cet affichage doit être maintenu tant que BP1 reste enfoncé
« Tant que finie », cet affichage doit apparaître dès que BP1 est relâché

5.4) Développez en C le programme qui correspond à l'ordinogramme exposé en 5.3)

En exploitant bien la fonction lcd_goto(), il est possible de faire défiler un texte de gauche à droite sur l'écran. Pour cela il faut utiliser une variable (nommée *pos* par exemple) qui servira de paramètre à la fonction lcd_goto() et s'arranger pour que la valeur de la variable s'incrémente au sein d'une boucle. Il est également possible de s'arranger pour que le texte qui défile recommence au début de l'écran en faisant un test sur la valeur de cette variable.

5.5) ☺ ☺ « Coucou » défile sur écran

Développez un programme qui fasse défiler « coucou » de gauche à droite sur écran tant que BP1 reste enfoncé. Arrangez-vous également pour que le défilement puisse recommencer au début dés que le texte sort de l'écran.

A partir de maintenant nous allons prendre soin de développer tous les programmes en utilisant une boucle infinie. Dans ces conditions nous n'allons plus mettre l'instruction « while(1) ; » (qui signifie « stop ici ») à la fin du programme mais placer une boucle infinie dont la syntaxe en C est :

```
while(1)
        {
        //tout ce qui se trouve ici est répété à l'infini
         }
```

Tout ce qui se trouve entre { et } sera répété indéfiniment.

5.6) Exposez un ordinogramme qui exploite la notion de boucle infinie de manière à faire clignoter la LED1 en permanence. La fréquence de clignotement doit être de 5Hz.

5.7) Développez en C le programme qui correspond à l'ordinogramme exposé en 5.6)

5.8) En conservant le principe de la boucle infinie, exposez un ordinogramme qui permet de faire clignoter toutes les LEDS tant que BP1 est enfoncé.

5.9) Développez en C le programme qui correspond à l'ordinogramme exposé en 5.8)

Pour les deux exercices suivants, rappelez-vous des opérateurs logiques sur entiers :
ET ↔ && OU ↔ || NON ↔ !

5.10) ☺ ☺ Cligno2-3 tant que BP 1 et 2
Développez un programme en C qui gère les clignotements des LEDS 2 et 3 tant que BP1 et BP2 sont tous les deux enfoncés.

5.11) ☺ ☺ Cligno1-8 tant que BP 1 ou 2
Développez un programme en C qui gère les clignotements des LEDS 1 et 8 tant que BP1 ou BP2 est enfoncé (il s'agit d'un OU non exclusif).

5.12) Exposez un ordinogramme qui permet d'afficher le nombre de fois que BP1 a été enfoncé.

Comment afficher sur écran le contenu d'une variable ?

Pour pouvoir afficher le contenu d'une variable sur l'écran de la carte info, nous allons utiliser une fonction nommée « sprintf() ». Le but de la fonction « sprintf() » est de placer dans un tableau tous les caractères qui devront être affichés (y compris les chiffres qui donnent la valeur de la variable).
Le contenu du tableau de caractères pourra alors être affiché à l'aide de la fonction « lcd_puts() » .

Il faut savoir que si on a une variable nommée *var* dont on veut afficher le contenu, il faudra afficher les chiffres qui constituent le nombre contenu au sein de cette variable. Ces chiffres sont aussi considérés comme des caractères, ils devront être stockés dans un tableau de caractère avant d'être affichés.
Comme montré à la page suivante, on utilise un tableau de caractères appelé L*igne1* et on inclut la librairie <stdio.h> pour pouvoir utiliser la fonction sprintf.

Exemple didactique d'utilisation du sprintf(…) :

```
#include        <stdio.h>
…
char Ligne1[16];
void main(void)
{
int var =89;
…
sprintf(Ligne1," voila : %2d bla", var);
lcd_goto(0);
lcd_puts(Ligne1);       // sera affiché :  « voila : 89 bla »
…
```

Suggestion pour mettre en oeuvre le sprintf(…) :

Faites le test suivant (exercice "My first sprintf"):
Déclarez une variable de type entière puis affectez-la d'une valeur. Ensuite affichez le contenu de cette variable sur la première ligne de l'écran. Pour cela il faut utiliser une fonction sprintf(…) mais alors il faut utiliser la librairie <stdio.h>. La fonction sprintf(…) a besoin d'un tableau de caractères.

5.13) En mettant en œuvre la fonction sprintf(), basez-vous sur l'ordinogramme exposé en 5.12) pour développer le programme équivalent en C.
Référez-vous aux encadrés précédents.

Série 6 : Exemples de programmes

6.1) Après voir essayé de comprendre, recopiez le programme « chrono de base » de votre syllabus de théorie (Fig.13 p.22), testez-le et analysez-le.

6.2) En vous basant sur le programme du chrono de base (exercice 6.1), essayez d'exposer par vous-même l'ordinogramme correspondant.

6.3) ☺ ☺ ☺ Amélioration chrono de base

Essayez de prévoir une amélioration du chrono de base de manière à :

- Permettre de choisir le moment de démarrage du chrono à l'aide de BP1, on doit attendre un enfoncement de BP1.
- Gérer l'incrémentation des minutes.
- Permettre de stopper le chrono à l'aide du BP4 (le chrono fonctionne donc tant que BP4 est relâché)

6.4) Après voir essayé de comprendre, recopiez le programme « carré d'un nombre » de votre syllabus de théorie (Fig.14 p.23), testez-le et analysez-le.

6.5) En vous basant sur le programme du « carré d'un nombre » (exercice 6.4), essayez d'exposer par vous-même l'ordinogramme correspondant.

Exercice récapitulatif, gare de train

1°) Vous devez afficher les lignes ci-dessous en affichant d'abord la 1ère ligne (puis attendre 2s) et ensuite la 2ème ligne (puis encore attendre 1s) :

Q	1			Q	2		N	0		Q	3	*	R	E	T
							X	X				*	B	P	4

Q1 représente le quai n°1 d'une gare, Q2 le quai n°2, Q3 le quai n°3.
XX représente le numéro du train entrant en gare.

2°) Toutes les dix secondes, un train doit entrer en gare.
XX doit indiquer le numéro du train entrant.

Q	1			Q	2		N	0		Q	3	*	R	E	T
Y	Y			Z	Z		X	X		W	W	*	B	P	4

Le train entrant doit toujours se garer sur la voie libre portant le plus petit numéro.
Si aucune voie n'est libre, les trains doivent arrêter d'entrer en gare (le compteur XX doit s'arrêter).

La pression sur le BP1 doit faire partir le train du quai n°1, La pression sur le BP2 doit faire partir le train du quai n°2, la pression sur le BP3 doit faire partir le train du quai n°3

Q	1			Q	2		N	0		Q	3	*	R	E	T
Y	Y			Z	Z		X	X		W	W	*	B	P	4

3°) Dés que le vingtième train est entré en gare, les trains doivent arrêter d'arriver, et les trains présents doivent partir les uns derrière les autres à intervalle de 5 secondes.

T	R	A	I	N	S						*	R	E	T	.
A	R	R	I	V	E	S		!	!		*	B	P	4	

Essayer de faire en sorte pour que lorsqu'on se trouve au niveau du point 2° ou 3°, on puisse grâce au BP4, revenir au point 2° dans la situation de départ (les quais sont vide et c'est le train n° 1 qui est sur le point de se présenter).

Série 7 : La « répéter … tant que »

Comme pour la série d'exercices précédente, ainsi que pour tous les exercices qui suivent, vous devez mettre en œuvre une boucle infinie au sein de votre programme principal tel que présenté Fig.12 p.21 du syllabus de théorie.
Vous devez prendre soin de mettre dans la boucle infinie tout ce qui est susceptible de se répéter.

7.1) Exposez un ordinogramme qui utilise une « répéter…tant que » de manière à avoir :
- Apres la carte initialisée, on veut au moins un clignotement de toutes les LEDS.
- Le clignotement de toutes les LEDS doit être répété tant que BP1 reste enfoncé.
- Lorsque le clignotement des LEDS est terminé (soit BP1 n'a pas été enfoncé, soit il vient d'être relâché) il faut deux clignotements de la LED1 puis un délai de 2s.
- Les délais pour tous les clignotements calculés pour avoir 5Hz.

7.2) Développez en C le programme qui correspond à l'ordinogramme exposé en 7.1)

7.3) Exposez un ordinogramme équivalent à l'exercice 7.1) mais qui est basé sur la structure d'une « tant que » (il s'agit d'un programme didactique qui met en œuvre

une « répéter …tant que » à l'aide d'une « tant que).

7.4) Développez en C le programme qui correspond à l'ordinogramme exposé en 7.3)

7.5) ☺ ☺ Gestion BP1 appuyé/enfoncé avec « répéter…tant que »

Affichez au départ sur écran

A	P	P	U	Y	E	Z		S	U	R		B	P	1	

Arrangez-vous ensuite pour attendre l'enfoncement de BP1 à l'aide d'une « répéter …tant que » vide. La syntaxe est

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

Dès que BP1 est enfoncé, affichez

R	E	L	A	C	H	E	Z		B	P	1				

Arrangez-vous ensuite pour attendre le relâchement de BP1 (toujours à l'aide d'une « répéter …tant que » vide).

7.6) ☺ ☺ Gestion BP1 appuyé/enfoncé avec « tant que »

Faites exactement le même exercice que le 7.5) mais en utilisant une « tant que » pour attendre l'enfoncement de BP1 ainsi que pour l'attente du relâchement de BP1.

7.7) ☺ ☺ Gestion BP1 appuyé/enfoncé avec « répéter…tant que » abrégée

F aites exactement le même exercice que le 7.5) et le 7.6) mais en utilisant la forme abrégée de la « répéter…tant que ».

Astuce pour attendre par exemple que BP1 et BP2 soient tous les deux enfoncés (cf 7.8) :

Il faut se dire qu'on ne fait rien tant que la condition n'est pas remplie ou, autrement dit, tant que la NON condition est vraie. Pour cela on utilise l'opérateur logique NON (qui est '!' en C) devant la condition souhaitée (BP1 et BP2 dans le cas présent).

7.8) ☺ ☺ Attente que BP1 et BP2 tous les deux enfoncés

Développez un programme qui a pour but d'initialiser la carte, allumer toutes les LEDS puis attendre que les boutons BP1 et BP2 soient tous les deux enfoncés avant de passer à la suite et gérer un clignotement permanent de toutes les LEDS.

Série 8 : La boucle « pour »

8.1) Exposez un ordinogramme qui utilise une « tant que » pour faire 10 clignotements de la LED1 après l'initialisation. Une fois les clignotements terminés il faut allumer toutes les LEDS avant de rentrer dans la boucle infinie.
Pour cet exercice la boucle infinie peut rester vide.

8.2) Développez en C le programme qui correspond à l'ordinogramme exposé en 8.1)

8.3) ☺ Séquence 3-5

Mettez en œuvre des boucles « for » pour avoir en permanence la séquence suivante :
3 clignotements de la LED1 (f=10Hz) -5 clignotements de la LED2 (f=10Hz).

8.4) ☺ Séquence 4-3-2

Mettez en œuvre des boucles « for » pour avoir en permanence la séquence suivante :
4 clignotements de la LED1 (f=10Hz) -3 clignotements de la LED2 (f=10Hz) et 2 clignotements de la LED3 (f=10Hz).

8.5) ☺ ☺ BP influence nombre clignotements
Il faut qu'au départ on ait, en boucle, une séquence 3-2, c'est-à-dire :
3 clignotements de la LED1 (f=10Hz) -2 clignotements de la LED2 (f=10Hz).
Il faut maintenant pouvoir incrémenter, à l'aide du dernier bouton-poussoir, le nombre de clignotement de la LED2. Prévoyez un délai anti-rebonds après une détection d'enfoncement.

8.6) ☺ ☺ Séquence 3-2-3-2-3-2-5
Mettez en œuvre la notion de boucle « for » pour avoir en permanence la séquence suivante :
3 fois de suite : 3 clignotements de la LED1 (f=10Hz) -2 clignotements de la LED2
puis 5 clignotements de l'ensemble des 8 LEDS (f=10Hz)
Il est suggéré d'avoir des « for » imbriqués.

8.7) ☺ ☺ ☺ Calcul puissance quelconque
Essayez de prévoir une amélioration du « carré d'un nombre » (voir Fig.14 p.23) de manière à pouvoir calculer une puissance quelconque. BP1 permet d'incrémenter la base alors que BP2 permet d'incrémenter l'exposant.

Série 9 : Les types en C

9.1) Après avoir déterminé le nombre de combinaisons différentes qui existent avec 3 bits, donnez l'ensemble des combinaisons possibles.

9.2) Combien il y a-t-il de combinaisons différentes avec 9 bits ?
Quel est l'ensemble des nombres entiers positifs représentables ?

9.3) Quelle est, en utilisant le compilateur PICC18, la plus grande valeur entière représentable en utilisant un type qui fait partie de la famille des entiers ?
Citez le type utilisé et justifiez votre réponse.

9.4) On doit déclarer une variable de la famille des entiers qui contiendra une valeur toujours comprise entre -200 et +200, quel type doit-on utiliser en C ?

9.5) Quels sont les nombres représentables avec le type **signed long** (compilateur PICC18)?

Série 10 : Les opérateurs logiques

10.1) Simplifiez l'expression : $a + ab$ Vous pouvez mettre en évidence

10.2) Simplifiez l'expression : $a \times (a + b)$ Vous pouvez distribuer

10.3) Simplifiez l'expression : $(a + b) \times (a + \bar{b})$ Vous pouvez distribuer

10.4) Simplifiez l'expression : $(a+b)\times(a+c)$ Distribuez puis mettez en évidence

10.5) Simplifiez l'expression : $(a+b+c)\times(a+\overline{b}+c)\times(a+\overline{b}+\overline{c})$

10.6) Montrez à l'aide de l'algèbre de Boole que les conditions :
!((BP1==1) && (BP2==1)) , ((BP1==0) || (BP2==0))
sont équivalentes. Donnez-en ensuite la signification.

10.7) En considérant les déclarations ci-dessous, donnez le contenu de la variable *a* pour les cas d'affectations donnés :
char a, op1=0b11110000, op2=0b10101010, op3=0b10000000, op4=0b011111111 ;
10.7.1) a = op1 & op2 ;
10.7.2) a = op1 & op3 ;
10.7.3) a = op1 & op4;
10.7.4) a = op2 | op1 ;
10.7.5) a = op2 | op3 ;
10.7.6) a = op2 | op4 ;
10.7.7) a = op2 | 0b11111111 ;
10.7.8) a = op2 & 0b11111111 ;
10.7.9) a = op2 | 0b00000000 ;
10.7.10) a = op2 & 0b00000000 ;

Série 11 : Applications des opérateurs

Les masques binaires

Dans certains cas il est utile de pouvoir vérifier (ou tester) si un bit particulier d'une variable est à '0' ou à '1'. Dans ce cas on ne peut pas simplement tester la valeur de cette variable car sa valeur ne dépend pas uniquement de l'état du bit qui nous intéresse mais également de l'état des autres bits qui constituent cette variable. La démarche est alors d'utiliser un masque bien choisi afin d'obtenir un motif binaire qui ne dépendra que du bit dont on veut vérifier l'état. Pour mettre en œuvre la notion de masque nous allons manipuler « PORTA » comme une variable et l'associer à un masque afin de pouvoir tester l'état des boutons-poussoirs sans utiliser RA0, RA1,… ni les 'define' associés BP1, BP2,…
Il faut également prévoir qu'on peut avoir plusieurs BP enfoncés en même temps.

11.1) Sans utiliser l'opérateur ET bit à bit (&) :
Affichez sur la 1ère ligne de l'écran :
« BP1 ENFONCE» ou « BP1 RELACHE » suivant que BP1 est enfoncé ou non.
Et sur la 2ème ligne :
« BP2 ENFONCE» ou « BP2 RELACHE » suivant que BP2 est enfoncé ou non.

11.2) Sans utiliser l'opérateur OU bit à bit (|)
Affichez sur la 1ère ligne de l'écran :
« BP3 ENFONCE» ou « BP3 RELACHE » suivant que BP3 est enfoncé ou non.
Et sur la 2ème ligne :
« BP4 ENFONCE» ou « BP4 RELACHE » suivant que BP4 est enfoncé ou non.

11.3) Ici nous allons essayer de lire et modifier l'état des LEDS individuellement sans utiliser

RB7, RB6,… ni les 'define' associés LED1, LED2,…
Nous pouvons cependant utiliser RA0, RA1 (ou les 'define' associés BP1, BP2,…) afin de faciliter les tests individuels des boutons-poussoirs.
Il s'agit de pouvoir modifier l'état des leds LED1 à LED4 à l'aide des boutons-poussoirs respectifs BP1 à BP4.
ATTENTION, il s'agit de pouvoir modifier l'état des leds, il faut donc vérifier au préalable si la led à modifier est allumée ou non afin de modifier correctement son état.

11.4) Même exercice que le 11.3) mais avec une contrainte supplémentaire :
On ne peut pas manipuler individuellement les boutons-poussoirs à l'aide de RA0, RA1,… ou des 'define' associés BP1, BP2,…

Les décalages binaires

Les opérateurs << et >> sont respectivement les opérateurs de décalage à gauche et à droite, ces opérateurs doivent être accompagnés d'un nombre entier pour indiquer de combien de crans on souhaite faire le décalage.
Il faut savoir que ces opérateurs ne modifient pas la valeur des opérandes, pour avoir une modification il faut une affectation. Autrement dit si vous écrivez « PORTB >> 1 ; » le contenu de PORTB ne sera pas modifié. Pour modifier PORTB il faut écrire par exemple : « PORTB = PORTB >> 1 ; ». Comme il y a une affectation, PORTB peut être modifié.

11.5) ☺ ☺ Chenillard avec décalages
On prétend que si on affecte le motif binaire 0b10000000 au PORTB (les LEDS), un décalage dans le temps du motif binaire de PORTB peut entraîner un chenillard. Basez vous sur ce principe pour gérer des trajets chenillards de gauche à droite avec un opérateur de décalage.

11.6) ☺ ☺ Chenillard droite à gauche avec décalages
Même chose qu'à l'exercice 11.7) mais avec un chenillard qui se déplace de droite à gauche.

11.7) ☺ ☺ Chenillard aller/retour avec décalages
Effectuez des chenillards aller/retour en boucle.

11.8) Dans la librairie lcd (au sein du fichier lcd.c), il y a la fonction suivante :

```
void lcd_write(unsigned char c)                    {
        unsigned char data;
        data=PORTB;
        PORTB = (PORTB&0x0f)|(c&0xf0);
        LCD_STROBE;
        PORTB = (PORTB&0x0f)|((c&0x0f)<<4);
        LCD_STROBE;
        delay_us(50);
        PORTB=data;
}
```

Sachant que « c » est une variable de 8 bits qui doit être envoyée sur l'écran lcd et que l'écran est en mode 4 bits, analysez et décrivez la manière dont les 8 bits seront transmis à l'écran.

Série 12 : Les fonctions, notions de base

12.1) ☺ Fct 1 clignotement 5Hz

Implémentez une fonction qui se charge de provoquer un clignotement de toutes les leds avec 100 ms comme délai. Faites ensuite un appel de cette fonction au sein de la boucle infinie, il devrait alors avoir un clignotement en permanence.

12.2) ☺ ☺ Fct 1 clignotement 5Hz avec pause

Réutilisez la même fonction que celle implémentée à l'exercice 1 mais en prenant soin de mettre un délai de 1 s. après l'appel de fonction. On verra alors le clignotement s'interrompre.

12.3) ☺ ☺ 2 appels Fct cligno puis pause

Réutilisez la même fonction que celle implémentée aux exercices 1 et 2 et, tout comme pour l'exercice 2, mettez un délai de 1 s. après l'appel de fonction. La différence ici est que l'on va, au sein de la boucle infinie, appeler la fonction 2 fois de suite avant de mettre le délai de 1 s. On va alors voir chaque fois deux clignotements avant chaque interruption.

12.4) ☺ ☺ ☺ Nb clignos en paramètre (séquence 6-4-2)

L'idée ici est de pouvoir spécifier, au moment de l'appel le nombre de clignotements que l'on veut. Il faut donc une fonction avec un paramètre pour le nombre de clignotements.
On va ensuite appeler cette fonction au sein de la boucle infinie de manière à avoir la séquence suivante qui se répète :
6 clignotements – délai 1s – 4 clignotements – délai 1s – 2 clignotements – délai 1s

12.5) ☺ Fct 1 trajet chenillard

Implémentez une fonction qui se charge de provoquer un trajet chenillard. Faites ensuite un appel de cette fonction au sein de la boucle infinie, il devrait alors avoir un chenillard en permanence.

12.6) ☺ ☺ chenillard avec pause

Réutilisez la même fonction que celle implémentée à l'exercice 6 mais en prenant soin de mettre un délai de 1 s. après l'appel de fonction. On verra alors le chenillard s'interrompre.

12.7) ☺ ☺ ☺ Nb chenillard en paramètre (séquence 3-pause)

L'idée ici est de pouvoir spécifier, au moment de l'appel le nombre de trajets chenillard que l'on veut. Il faut donc une fonction avec un paramètre. On va ensuite appeler cette fonction au sein de la boucle infinie de manière à avoir la séquence suivante qui se répète :
3 trajets chenillards – délai 1s

12.8) ☺ ☺ ☺ Somme et nb enfoncements

Implémentez une fonction avec deux paramètres entiers et qui donne comme valeur de retour la somme de ces deux nombres (donc également un entier). Utilisez ensuite cette fonction de sorte à afficher les lignes telles que ci-dessous :

B	P	1		B	P	2			S	O	M	M	E		
X	X	X		Y	Y	Y					Z	Z	Z		

XXX est le nombre de fois que BP1 a été enfoncé, YYY est le nombre de fois que BP2 a été enfoncé et ZZZ=XXX+YYY.

12.9) ☺ ☺ ☺ Nb chenillard en paramètre et choix G/D

On veut une fonction avec deux paramètres : un paramètre qui permet d'indiquer le nombre de trajets chenillards qui se répètent (comme pour l'exercice 7) et un autre paramètre de type « char » qui permet de spécifier si le chenillard va de gauche à droite (valeur 0) ou de droite à gauche (valeur autre que 0). Testez ensuite le bon fonctionnement de cette fonction en provoquant la séquence : 2 trajets chenillard à gauche - 3 trajets chenillard à droite

12.10) ☺ ☺ ☺ clignos f variable

On veut une fonction qui provoque un clignotement de toutes les leds mais avec le nombre de millisecondes entre les changements d'états qui soit paramétrable (un entier). Utilisez ensuite cette fonction au sein de la boucle infinie de manière à avoir les leds qui clignotent de plus en plus vite. Vous devez pour ça utiliser une variable dont la valeur variera de 1000 à 50 en se décrémentant par pas de 50.Une fois arrivé à 50 on doit réaffecter la valeur 1000 à cette variable pour que la fréquence progressive puisse reprendre.

12.11) ☺ ☺ ☺ clignos f variable avec nb BP

Réutilisez la fonction de l'exercice 10 et utilisez la au sein de votre programme pour avoir clignotement de toutes les leds à une fréquence proportionnelle au nombre de boutons-poussoirs enfoncés. Maintenez néanmoins une fréquence minimale de 0,5 Hz dans le cas particulier où aucun bouton-poussoir n'est enfoncé.

12.12) ☺ ☺ ☺ Max et nb enfoncements

Implémentez une fonction avec deux paramètres entiers et qui donne comme valeur de retour le maximum de ces deux nombres (donc également un entier). Utilisez ensuite cette fonction de sorte à afficher les lignes telles que ci-dessous :

B	P	1		B	P	2					M	A	X		
X	X	X		Y	Y	Y					Z	Z	Z		

XXX est le nombre de fois que BP1 a été enfoncé, YYY est le nombre de fois que BP2 a été enfoncé et ZZZ est le maximum parmi XXX et YYY.

12.13) ☺ ☺ ☺ Carré du min d'enfoncements

Implémentez deux fonctions :

Une fonction « min » qui renvoie le minimum de deux entiers.

Une fonction « carre » qui renvoie le carré d'un nombre entier.

Afficher ensuite les lignes telles que ci-dessous :

B	P		B	P		M	I	N			C	A	R	R	E
X	X		Y	Y			Z	Z				U	U	U	U

XX est le nombre de fois que BP1 a été enfoncé, YY est le nombre de fois que BP2 a été enfoncé, ZZ est le minimum parmi XX et YY et enfin UUUU=$(ZZ)^2$.

Série 13 : Les bits indicateurs d'état

13.1) ☺ ☺ Menu principal + 1 sous-menu

Il faut qu'au départ il soit affiché :

M	E	N	U		P	R	I	N	C	I	P	A	L		
	B	P	1		P	O	U	R		C	L	I	G	N	O

Et qu'en même temps toutes les leds sont allumées.

Il s'agit du menu principal qui est susceptible d'être répété, il se trouve donc dans la boucle infinie. Cependant il est possible que ce ne soit pas le menu principal qui est en cours, il y a aussi un sous-menu. On peut alors utiliser un « flag » nommé par exemple *menu_pr_en_cours* qui indique si c'est le menu principal qui est en cours. Lorsque ce flag contient la valeur 1 (vrai) ça signifie que ça doit être le menu principal, autrement ça doit être le sous-menu.
Ce qui est très pratique avec les flags c'est qu'il suffit de tester pour savoir ce qu'il faut faire (par exemple : if (*menu_pr_en_cours*= =1)…) mais cela requiert que la valeur du flag soit toujours représentative de la réalité. Le programmeur doit alors prendre soin de bien initialiser le flag et de le mettre à jour régulièrement.

A partir du menu principal, BP1 doit pouvoir influencer le menu en cours, on doit avoir :

S	O	U	S	-	M	E	N	U				*	R	E	T
C	L	I	G	N	O							*	B	P	4

Et qu'en même temps la 1ère led doit clignoter.
Un enfoncement de BP4 nous ramène au menu principal et l'allumage de toutes les leds.

13.2) ☺ ☺ ☺ Menu principal + 1 sous-menu mais même BP

Il s'agit exactement le même exercice que le 1.) sauf que pour le menu principal c'est BP4 qui permet d'atteindre le sous-menu. On constate que c'est donc le même bouton BP4 qui modifie le menu en cours.

13.3) ☺ ☺ ☺ Gestion d'un flag nommé *tour_chenillard*

Imaginons un programme qui fait déplacer la led allumée vers la droite à l'aide de BP1. Quand on arrive au bout des 8 leds, il faut un trajet chenillard avant de reprendre le procédé. Utiliser une variable « flag » nommée *tour_chenillard* qui indique si c'est à ce tour que le chenillard doit avoir lieu. Le « flag » doit être initialisé, régulièrement mis à jour et testé lorsque nécessaire.

13.4) ☺ ☺ ☺ Chenillard auto et manu

Affichez dés le départ (et une seule fois)

		A	U	T	O			M	A	N	U				
			B	P	2				B	P	3				

BP1 et BP2 permettent tous les deux de rentrer dans le menu suivant (menu chenillard) mais BP2 pour un chenillard auto et BP3 pour un chenillard manuel. Dans la mesure où l'on choisit MANU, c'est BP1 qui doit permettre de faire avancer le chenillard manuellement.
Utilisez une variable nommée « manu » qui indique si on est en mode manuel ou non.

13.5) ☺ ☺ ☺ ☺ Menu principal + 2 sous-menu2

Imaginez une amélioration du 4.) en laissant la possibilité de revenir au menu principal. Il faut alors utiliser un 2ème « flag » tel que *menu_pr_en_cours*.

Série 14 : Les entiers indicateurs d'état (gestion de menus)

14.1) ☺ ☺ Menu départ+ 2 sous-menus

Il faut qu'au départ il soit affiché :

B	P	1		B	P	2									
	1				2										

Le bouton-poussoir 1 permet alors de sélectionner le sous-menu1, tandis que le bouton-poussoir permet de sélectionner le sous-menu2. Si on sélectionne le sous-menu1 on aura :

S	O	U	S	-	M	E	N	U	1			*	R	E	T
												*	B	P	4

On voit que le numéro du sous-menu est affiché à la 1ère ligne. Le bouton-poussoir 4 permet de revenir à l'affichage de départ. Bien sûr si au départ on avait choisi le sous-menu2, c'est celui-ci qui aurait été indiqué en ligne1.

14.2) ☺ ☺ Menu départ+ 2 sous-menus dont cligno

Il faut qu'au départ il soit affiché :

B	P	1		B	P	2		B	P	3					
C	L	I	G	N	O				3						

Les boutons-poussoirs 1 et 2 permettent tous les deux de rentrer dans le sous-menu cligno au sein duquel il y a clignotement de toutes les leds. Il doit aussi être affiché :

C	L	I	G	N	O							*	R	E	T
												*	B	P	4

Il y a aussi le sous-menu3 qui aurait pu être choisi mais là rien ne se passe ; il est simplement écrit en ligne1 le nom de ce sous-menu.

Peu importe le sous-menu dans lequel nous nous trouvons, il est possible de revenir à l'affichage de départ à l'aide du bouton-poussoir 4.

14.3) ☺ ☺ Arborescence deux niveaux

L'idée est d'avoir deux sous-menus (1 et 2) qui permettent chacun d'accéder à deux autres « sous sous-menus ». Il y a donc les « sous sous-menus » nommés 1.1, 1.2, 2.1 et 2.2.

Il faut qu'au départ il soit affiché :

B	P	1		B	P	2		D	E	P	A	R	T		
	1				2										

Si on appuie par exemple sur BP1, on doit arriver dans le sous-menu 1, le numéro du menu doit alors apparaître au milieu de la 1ère ligne comme ci-dessous :

B	P	1		B	P	2		1				*	R	E	T
1	.	1		1	.	2						*	B	P	4

On remarque également que la 2ème ligne propose les « sous sous-menus » disponibles.

Evidemment si on avait appuyé sur BP2 on aurait dû arriver dans le sous-menu 2 (et c'est alors celui-ci qui aurait été indiqué au niveau de la 1ère ligne). Les « sous sous-menus » disponibles auraient alors été 2.1 et 2.2.

A partir d'un sous-menu il faut pouvoir accéder à un « sous sous-menus » à l'aide de BP1 ou de BP2. On pourrait alors avoir un affichage tel que ci-dessous :

S	O	U	S									*	R	E	T
S	O	U	S	-	M	E	N	U	1	.	2	*	B	P	4

Il faut bien sûr adapter l'affichage au « sous sous-menus » sélectionné.

Il faut aussi prévoir de revenir à l'affichage précédent à l'aide du dernier bouton-poussoir, et ce à tout moment.

14.4) Exposez l'ordinogramme qui correspond à l'exercice 14.1)

14.5) Exposez l'ordinogramme qui correspond à l'exercice 14.2)

14.6) Exposez l'ordinogramme qui correspond à l'exercice 14.3)

Exercice récapitulatif, Chenillards

1°) Il faut qu'au moment du reset de programme, il soit indiqué sur l'écran :

E	X	A	M	E	N		D	'	I	N	F	O		A	.
P	A	T	I	E	N	T	E	Z		5	s				

Tout en laissant les inscriptions sur l'écran, vous devez ensuite faire en sorte que la 1ère led clignote 5 fois avec un intervalle de 1s entre 2 clignotements consécutifs.
Réalisez qui précède l'affichage et le clignotement qui précède au sein d'une fonction.

2°) Une fois les clignotements terminés vous devez afficher sur écran :

C	H	E	*	C	P	T	*	N	B	R	*	R	E	T	.
B	P	1	*	B	P	2	*	B	P	3	*	B	P	4	

Gérez cet affichage au sein d'une fonction

3°) Ensuite à partir de ce menu principal, vous avez 4 possibilités :

a.) Le BP1 vient d'être enfoncé, vous devez afficher :

S	O	U	S	-	M	E	N	U		1	*	R	E	T	.
C	H	E	N	I	L	L	A	R	D		*	B	P	4	

Et il doit apparaître un chenillard au niveau des leds. La seule façon de sortir de ce sous-menu est d'appuyer sur le BP4, dans quel cas revenez au menu principal.

b.) Le BP2 vient d'être enfoncé, vous devez afficher :

S	O	U	S	-	M	E	N	U		2	*	R	E	T	.
B	P	1		X	X		F	O	I	S	*	B	P	4	

Où XX correspond au nombre de fois que BP1 a été enfoncé au sein de ce menu. La seule façon de sortir de ce sous-menu est d'appuyer sur BP4 (on retourne alors au menu principal).

c.) Le BP3 vient d'être enfoncé, vous devez afficher :

S	O	U	S	-	M	E	N	U		3	*	R	E	T	.
X		B	P		E	N	F	.			*	B	P	4	

Où X correspond au nombre de BP enfoncés parmi les BP1,2 et 3. La seule façon de sortir de ce sous-menu est d'appuyer sur le BP4, dans quel cas revenez au menu principal.

d.) Le BP4 vient d'être enfoncé, vous devez :

Revenir au tout 1er affichage et faire en sorte que le programme recommence en 1°, il faut évidemment qu'ensuite on puisse arriver dans le menu principal et sélectionner un sous-menu.

Essayer de faire en sorte que les points a.), b.) et c.) ci-dessus puissent être réalisés par des fonctions.

Suggestion :

Lorsque vous êtes en train de gérer le chenillard (probablement dans une boucle) et que vous revenez au menu principal, vous devez interrompre votre chenillard ; pour ce faire, vous pouvez utiliser l'instruction « break ; » qui permet de sortir de la boucle en cours.

Exercice récapitulatif, Course auto

1°) Il faut qu'au moment du reset les leds 3 et 4 clignotent 10 fois avec un intervalle de 0,5s entre 2 clignotements consécutifs. Le nombre de clignotement doit être paramétrable, c'est-à-dire que s'il faut modifier pour avoir 20 clignotements, il n'y a qu'une seule chose à changer dans le programme.

2°) A la fin des clignotements de leds, vous devez afficher les lignes ci-dessous en affichant d'abord la 1ère ligne (puis attendre 2s) puis la 2ème ligne (puis encore attendre 1s) :

E	X	A	M	E	N		D	'	I	N	F	O		A	.
C	O	U	R	S	E		V	O	I	T	U	R	E	S	

3°) A la fin du point 2°, vous devez afficher, à l'aide d'une fonction nommée affichage_choix_nb_tours( ), les lignes suivantes :

1	0			*		1	5			*		2	0		
B	P	1		*		B	P	2		*		B	P	3	

4°) Vous devez ensuite mémoriser le choix précédent (10, 15 ou 20 tours selon le BP enfoncé) et afficher les lignes ci-dessous où XX correspond au nombre de tours choisi, YY le nombre de tours effectués par l'équipe 1et ZZ le nombre de tours effectués par l'équipe 2. Une pression sur le BP2 doit incrémenter le nombre de tours effectués par l'équipe 1, et de même une pression sur le BP3 doit incrémenter le nombre de tours effectués par l'équipe 2.

X	X		E	Q	1		E	Q	2		*	R	E	T	.
			Y	Y			Z	Z			*	B	P	4	

5°) Dés qu'une des deux équipes atteint le nombre de tours choisi, vous devez afficher les lignes ci-dessous (où X correspond au numéro de l'équipe gagnante :

L	'	E	Q	U	I	P	E		X		*	R	E	T	.
A		G	A	G	N	E		!	!	!	*	B	P	4	

6°) Essayer de faire en sorte pour que lorsqu'on se trouve au niveau du point 4° ou 5°, on puisse grâce au BP4, revenir au point 3° pour recommencer à choisir un nombre de tours. Il faut évidemment pouvoir ensuite refaire jouer les équipes.

Série 15 : Langage de description algorithmique

15.1) En vous basant sur le programme du chrono de base (exercice 6.1), essayez d'exposer par vous-même le LDA correspondant.

15.2) En vous basant sur l'ordinogramme du « carré d'un nombre » (exercice 6.5), essayez d'exposer par vous-même le LDA correspondant.

15.3) Exposez le LDA qui correspond au programme de l'exercice récapitulatif : Gare de train

15.4) Exposez le LDA qui correspond au programme de l'exercice récapitulatif : Chenillards

15.5) Exposez le LDA qui correspond au programme de l'exercice récapitulatif : Course auto

Série 16 : Eléments supplémentaires

16.1) Imaginez une manière de tester l'instruction « break ; » au sein de la boucle infinie de votre programme principal. Il faut savoir que normalement on ne sort pas de la boucle infinie, il s'agit d'un exemple didactique.
Interrompez par exemple un clignotement permanent à l'aide d'une boucle infinie.

16.2) Développez un programme en C qui assure 50 clignotements de toutes les LEDS avant d'afficher un message de bienvenue, vous rentrez ensuite dans une boucle infinie vide. Cependant vous voulez prévoir la possibilité d'interrompre les clignotements à l'aide de BP1.

16.3) Vous devez déclarer, implémenter et utiliser une fonction nommée deux_exp() qui utilise une variable de type **char** comme paramètre et qui renvoie un type **int** comme valeur de retour. La fonction doit utiliser un tableau de 16 entiers qui sont les 16 premières puissances de 2 (en commençant à 0).
Au sein même de la fonction, déclarez un tableau et initialisez-le pour qu'il contienne les 16 premières puissances de 2 (en commençant par 1 à l'indice 0 du tableau).
Imaginez ensuite une manière de tester votre fonction avec la carte info.

16.4) Faites la même chose que l'exercice 16.3) mais sans déclarer de tableau au sein de la fonction, déclarez plutôt le tableau en global, juste avant la fonction main().

16.5) Faites la même chose que l'exercice 16.3) mais sans déclarer de tableau au sein de la fonction, déclarez plutôt le tableau en local au sein de la fonction main(). Vous devez alors imaginer une fonction qui utilise un pointeur comme paramètre. Vous pourrez alors passer le nom du tableau en paramètre ainsi que la valeur de l'exposant au moment de l'appel.

Manipulations liées au µC utilisé

A. Manipulation sur le Timer

Un timer peut travailler en compteur ou en timer.
En compteur, on compte les impulsions d'un signal externe.
En timer, le registre du timer peut s'incrémenter régulièrement de manière interne.

Le PIC18F2520 contient plusieurs modules timer/compteur : tmr0, tmr1, tmr2 et tmr3.
Ces modules sont différents de manière à choisir celui qui est le plus approprié à l'application.

1°) Etude du module tmr0 :

Il y a dans la RAM du PIC une zone réservée qui contient les SFR (les registres spéciaux).
Ces registres servent entre autre à configurer le PIC (timer, ports E/S, interruptions,…).

Pour le timer0 il y a :

TMR0L : 8 bits LSB pour la valeur actuelle du timer0
TMR0H : 8 bits MSB pour la valeur actuelle du timer0

T0CON : 8 bits (timer0 control) permettant d'agir sur certains aspects du tmr0
- Bit 7, TMR0ON : Pour lancer ou stopper le tmr0 (l'incrémentation)
- Bit 6, T08BIT : Pour travailler en 8bits ou 16bits
- Bit 5, T0CS : Pour choisir une incrémentation en interne ou non
- Bit 4, T0SE : Pour choisir le flanc actif pour l'incrémentation
- Bit 3, PSA : Pour utiliser ou non le PRESCALER (pré division)
- Bit2, Bit1, Bit0 ou T0PS2, T0PS1, T0PS0 : Pour configurer le PRESCALER (la pré division) Exemple (111 ↔ division par 256)

TMR0IE : tmr0 interrupt enable, 1 bit du registre INTCON (interruption control) pour autoriser ou non les interruptions timer0 (lors d'un dépassement de capacité de la valeur actuelle du timer0).

TMR0IP : tmr0 interrupt priority, 1 bit du registre INTCON2 pour configurer la priorité de l'interruption du tmr0.

TMR0IF : tmr0 interrupt flag, 1 bit du registre INTCON pour signaler s'il y a eu un dépassement de capacité de la valeur actuelle du timer0.

ATTENTION, le flag de dépassement de capacité (TMR0IF) doit être remis à 0 par soft !

Questions : Créez un projet à partir du dossier « timer0 sans interruptions en 16 bits » ou en recopiant le programme donné à la page suivante.

1.) Analysez toutes les lignes du programme et notez dans votre cahier des commentaires explicatifs sur chaque ligne.
2.) Comparez votre analyse et le comportement du programme.

3.) Que se passe-t-il lorsqu'on change la dernière ligne de init_tmr0() et que l'on met TMR0ON=0; Pourquoi ?

4.) Modifiez votre programme pour que la led clignote beaucoup plus lentement (2 secondes entre chaque changement d'état). Notez les changements apportés et expliquez comment vous avez procédé.

Le programme de base est ci-dessous :

```
void main (void)
{
init();
init_tmr0();
while (1)
        {
        RB6=1;
        TMR0=55536; TMR0IF=0; while(!TMR0IF);
        RB6=0;
        TMR0=55536; TMR0IF=0; while(!TMR0IF);
        }//fin du while(1)
}//fin du main
```

```
void init_tmr0()
        {
        T08BIT=0;
        T0CS=0;
        PSA=0;

/*      b2-b0 111 pour division par 256 (prescaler)
        5MIPS donc une instruction en 0,2 µs
        fréquence divisée par 256 donc une incrémentation chaque (0,2*256 µs)
        donc une incrémentation pour environ 0,05ms                    */
        T0PS2=1; T0PS1=1; T0PS0=1;

        TMR0IF=0;

/*      comme en 16 bits, capa max = 65 536
        en chargeant 65 536 - 10000 = 55 586        */
        TMR0=55536;

        TMR0ON=1;
        }
```

B. Manipulation sur les interruptions

Les interruptions sont très utiles en informatique embarquée.
Si on veut par exemple faire clignoter une led en permanence, avec une gestion classique le µC va passer tout son temps à la gestion du clignotement ; il va d'ailleurs passer la majorité de son temps à attendre.
L'idée est que sur une période de clignotement, le µC n'a réellement que deux choses à faire :
allumer la led et éteindre la led
On va alors utiliser les interruptions pour gérer le clignotement de la led. On va alors pouvoir utiliser le programme principal pour autre chose et gérer le clignotement de la led par interruption. Le programme principal va alors s'interrompre régulièrement pour aller changer l'état de la led.

Il peut il y avoir plusieurs sources d'interruptions (interne, timer, PWM, UART,…).

En général un module d'interruption peut être manipulé grâce à trois bits :

Exemple : TMR0IE pour autoriser ou non les interruptions timer0
TMR0IP pour configurer la priorité de l'interruption du tmr0.
TMR0IF pour signaler si le flag qui permet l'interruption = 1.

Pour les interruptions il y a :

INTCON : 8 bits (interrupt control) pour configurer les interruptions
- Bit 7, GIE : Pour autoriser ou non les interruptions (globalement)
- Bit 6, PEIE : Pour autoriser ou non les interruptions périphériques
- Bit 5, TMR0IE : Pour autoriser ou non les interruptions du timer0
- Bit 4, INT0IE : Pour autoriser ou non l'interruption externe INT0
- Bit 3, RBIE :
- Bit 2, TMR0IF : Indique s'il y a eu dépassement de capacité de tmr0
- Bit 1, INT0IF : Flag de l'interruption externe INT0
- Bit 0, RBIF :

INTCON2 : 8 bits (interrupt control) pour configurer les interruptions
Il y a entre autre le bit 2 lié au timer0 qui permet de choisir une haute ou une basse priorité pour l'interruption du timer0.

INTCON3 : 8 bits (interrupt control) pour configurer les interruptions

PIR et PIR2 : Registres liés aux flags d'interruptions
PIE1, PIE2 : Registres pour autoriser ou non les interruptions périphériques
IPR1 et IPR2 : Registres pour les priorités des interruptions périphériques
RCON : Registre (reset control register) qui contient entre autre le bit IPEN (interrupt priority enable) pour autoriser ou non plusieurs priorités.

Questions : Créez un projet à partir du dossier « timer0 avec interruptions en 16 bits » ou en recopiant le programme donné à la page suivante.

1.) Analysez toutes les lignes du programme et notez dans votre cahier des commentaires explicatifs sur chaque ligne.
2.) Comparez votre analyse et le comportement du programme.
3.) Mesurez la fréquence d'oscillation de RB0.
4.) Expliquez, en vous appuyant sur un calcul, pourquoi la fréquence d'oscillation de RB0 est celle mesurée.
5.) Que se passe-t-il lorsqu'à la fin de l'initialisation du timer, on met GIE=0; au lieu de GIE=1; Pourquoi ?
6.) Modifiez certaines lignes du programme pour que l'interruption du timer0 ait lieu à chaque milliseconde (exactement) et arrangez-vous pour que RB0 clignote alors (toujours par interruption) à 4 Hz. Expliquez toutes vos modifications.
7.) Faites clignoter la 1ère led dans la fonction main()(RB7) à l'aide de délais, cela affecte-t-il le clignotement de RB0 ? Pourquoi ?
8.) Utilisez une base de temps de 1 ms (une interruption du timer à chaque milliseconde) et exploitez cette base de temps pour faire clignoter la led RB0 à 5 Hz et RB1 à 10 Hz. Expliquez.

Le programme de base est ci-dessous :

```
void main (void)
{
init();
init_tmr0();

while (1)
        {
        }//fin du while(1)
}//fin du main
```

```
void init_tmr0()
        {
        IPEN=0;
        TMR0IE=1;
        TMR0IF=0;

        T08BIT=0;
        T0CS=0;
        PSA=0;
        T0PS2=1; T0PS1=1; T0PS0=1;
/*      b2-b0 110 pour division par 128 (prescaler)
        5MIPS donc une instruction en 0,2 µs
        fréquence divisée par 128 donc une incrémentation chaque (0,2*128 µs)
        donc une incrémentation pour environ 0,025ms                    */

        GIE=1;
        TMR0=55000;
        TMR0ON=1;
        }
```

```
interrupt mon_interrupt_tmr0()
        {
        TMR0IF=0;       //RAZ drapeau d'interruption
        TMR0=55000;
        nb++;
        if(nb>=2)
                {
                RB0=!RB0;
                nb=0;
                }
        }
```

C. Manipulation sur la gestion d'un servomoteur

Questions : Créez un projet à partir du dossier « gestion de 1 servo » ou en recopiant le programme donné à la page suivante.

1.) Analysez toutes les lignes du programme et notez dans votre cahier des commentaires explicatifs sur chaque ligne.
2.) Comparez votre analyse et le comportement du programme.
3.) Observez le signal qui se trouve sur RB4, expliquez.
4.) Que devient ce signal lorsqu'on appuie sur BP1 ? Pourquoi ?
5.) Connectez un servomoteur à la carte info, il y a trois fils :
VCC (rouge), GND (noir) et la commande (blanc ou jaune)

Pour les broches :

En enlevant l'écran, il y a deux coté de barrette sécable, le côté droit, la pin de gauche, c'est RB4, celle juste à côté RB5 puis RB6 et RB7.

La broche 2 de l'écran c'est VCC. La broche 5 (5ème pin) c'est GND (la masse).

Le programme de base est ci-dessous :

```
void main (void)
{
int a;
init();
init_tmr0();

while (1)
        {
        if(BP1==1)
                valeur_servo=20;
        else
                valeur_servo=100;
        }//fin du while(1)
}//fin du main
```

```
void init_tmr0()
        {
        IPEN=0;
        TMR0IE=1;
        TMR0IF=0;

        T08BIT=0;
        T0CS=0;
        PSA=0;
        T0PS2=0; T0PS1=0; T0PS0=1;
/*      b2-b0 001 pour division par 4 (prescaler)5MIPS donc une instruction en 0,2 µs
        fréquence divisée par 4 donc une incrémentation chaque (0,2*4 µs)
        donc une incrémentation pour environ 0,8 µs                     */

        //pour atteindre 20ms il faut 20000 µs / 0,8 µs donc 25000 incrémentations nécessaires
        TMR0=40536;

        TMR0ON=1;
        GIE=1;
        }
```

```
interrupt mon_interrupt_tmr0()
        {
        static int nb_incr_servo,i;

        TMR0IF=0;
        TMR0=40536;
        RB4=1;

        //on attend environ 1ms (les délais ici ne sont pas précis)
        delay_us(210);delay_us(210);delay_us(210);

        //on passe 10µs à chaque tour (la for prend 7µs
        // et 3 µs supplémentaire pour faire 10 µs
        for(i=0;i<valeur_servo;i++)
                {delay_us(3);}

        RB4=0;
        }
```

Quelques corrections d'exercices[1]

3.12.) ☺ Toutes leds

« prog.h », dans le cas d'un PIC18F2620

```
__CONFIG(1, HS&IESODIS&FCMDIS);
__CONFIG(2, BORDIS&PWRTEN&WDTDIS );
__CONFIG(3, PBANDIS);                                   //enlever cette ligne si on a un PIC18F2520
__CONFIG(4, LVPDIS&XINSTDIS&STVRDIS&LPT1DIS);
__CONFIG(5, UNPROTECT);
__CONFIG(6, WRTEN);                                     //enlever cette ligne si on a un PIC18F2520

extern void init(void);
```

« prog.c »

```
#include<pic18.h>
#include"prog.h"

#define  BP1 RA0
#define  BP2 RA1
#define  BP3 RA2
#define  BP4 RA3

#define  LED1 RB7
#define  LED2 RB6
#define  LED3 RB5
#define  LED4 RB4
#define  LED5 RB3
#define  LED6 RB2
#define  LED7 RB1
#define  LED8 RB0
#define  LEDS PORTB

void main(void)
{
init();
LEDS=0b11111111;        // LIGNES PROPRES
while(1) ;              // AU PRESENT EXERCICE
}//fin du main

/*************************************************************/
void init()
        {
        TRISA=0b11101111;
        ADCON1=0x0f;
        PORTA=0;
        TRISB=0b00000000;
        PORTB=0;
        TRISC=0b00000000;
        PORTC=0;
        }
```

[1] Par souci de nombre de pages et pour éviter d'inciter trop les élèves à regarder, nous avons volontairement omis la correction de certains exercices.

Remarques pour la suite :

- Le fichier « prog.h » reste identique à celui de la page précédente, seul « prog.c » doit être modifié pour correspondre à l'exercice concerné.
- Le début du fichier « prog.c » restera identique à celui de la page précédente (#include et #define) ainsi que l'implémentation de la fonction init( ). Les corrections des exercices suivants ne feront apparaître que l'essentiel (voir à la page précédente les « lignes propres au présent exercice »).

3.13) ☺ Led sur deux

```
LEDS=0b01010101;
while(1) ;
```

3.16) ☺ 2 cligno LED1

```
LED8=1;
delay_ms(100);
LED8=0;
delay_ms(100);
LED8=1;
delay_ms(100);
LED8=0;
delay_ms(100);
while(1) ;
```

Remarques pour la suite lors d'affichage sur écran :

- Les exercices qui affichent sur écran nécessitent d'inclure les fichier « lcd.h » et « lcd.c » au projet.
- Le fichier « prog.h » reste identique à celui des exercices précédents.
- Le début du fichier « prog.c » restera quasiment identique à celui des exercices précédents, il faut juste prendre soin de rajouter la ligne « #include "lcd.h" ».
- L'implémentation de la fonction init( ) au sein du fichier « prog.c » est quasiment identique à celle présentée précédemment, on prendra juste soin d'y insérer l'initialisation de l'écran au début. L'implémentation de la fonction init() devient alors :

```
void init()
    {
    TRISA=0b11101111;
    ADCON1=0x0f;
    PORTA=0;
    TRISB=0b00000000;
    PORTB=0;
    TRISC=0b00000000;
    PORTC=0;
    lcd_init();
    lcd_clear();
    PORTB=0;
    }
```

3.17) ☺ 3 alternances coucou - c'est moi

```
init();
lcd_goto(0x00);
lcd_puts("coucou");
delay_ms(1000);
lcd_goto(0x00);
lcd_puts("          ");
lcd_goto(0x40);
lcd_puts("c'est moi");
delay_ms(1000);
lcd_goto(0x40);
lcd_puts("          ");
lcd_goto(0x00);
lcd_puts("coucou");
delay_ms(1000);
lcd_goto(0x00);
lcd_puts("          ");

//suite de l'exercice ici à droite
```

```
//suite de l'exercice ci-dessous

lcd_goto(0x40);
lcd_puts("c'est moi");
delay_ms(1000);
lcd_goto(0x40);
lcd_puts("          ");
lcd_goto(0x00);
lcd_puts("coucou");
delay_ms(1000);
lcd_goto(0x00);
lcd_puts("          ");
lcd_goto(0x40);
lcd_puts("c'est moi");
delay_ms(1000);
lcd_goto(0x40);
lcd_puts("          ");
while(1) ;
```

Rem: La structure "pour" (voir plus loin) devrait être utilisée.

4.8) ☺ ☺ Toutes leds si BP

```
if((BP1==1)||(BP2==1)||(BP3==1)||(BP4==1))
        LEDS=0b11111111;
else
        LEDS=0b00000000;
while(1) ;
```

4.9) ☺ ☺ 4 1ères leds reflètent BP

```
LEDS=0b00000000;
LED1=BP1;
LED2=BP2;
LED3=BP3;
LED4=BP4;
while(1) ;
```

5.2)

```
while(BP1==1)
        {
        LED1=1;
        delay_ms(100);
        LED1=0;
        delay_ms(100);
        }
while(1) ;
```

5.5) ☺ ☺ « Coucou » défile sur écran

```
char pos=0;
init();
while(BP1==1)
        {
        lcd_goto(0x00);
        lcd_puts("          ");
```

```
        lcd_goto(pos);
        lcd_puts("coucou        ");
        delay_ms(200);
        pos=pos+1;
        if(pos==16)
                pos=0;
        }
while(1) ;
```

Remarque pour la suite :

A partir de maintenant nous allons prendre soin de développer tous les programmes en utilisant une boucle infinie. Dans ces conditions nous n'allons plus mettre l'instruction « while(1) ; » (qui signifie « stop ici ») à la fin du programme mais placer une boucle infinie dont la syntaxe en C est :

```
        while(1)
                {
                //tout ce qui se trouve ici est répété à l'infini
                 }
```

Tout ce qui se trouve entre { et } sera répété indéfiniment.

5.7)

```
while(1)
        {
        LED1=1;
        delay_ms(100);
        LED1=0;
        delay_ms(100);
        }//fin du while(1)
```

7.5) ☺ ☺ Gestion BP1 appuyé/enfoncé avec « répéter…tant que »

```
while(1)
        {
        lcd_goto(0x00);
        lcd_puts("APPUYEZ SUR BP1 ");
        do {}
        while(BP1==0);
        lcd_goto(0x00);
        lcd_puts("RELACHEZ BP1    ");
        do {}
        while(BP1==1);
        }//fin du while(1)
```

Remarque:

Dans certains cas on doit prévoir un système d'anti-rebonds des boutons-poussoirs

7.8) ☺ ☺ Attente que BP1 et BP2 tous les deux enfoncés

```
LEDS=0b11111111;
while ( !( (BP1==1)&&(BP2==1) ) );
while(1)
        {
        LEDS=0b00000000;
        delay_ms(100);
        LEDS=0b11111111;
        delay_ms(100);
        }//fin du while(1)
```

8.3) ☺ Séquence 3-5

```
char i;             // Les lignes de déclaration sont toujours au début (juste après le « { » du main()
init();
while(1)
{
for(i=0;i<3;i++)
        {
        LED1=1;
        delay_ms(50);
        LED1=0;
        delay_ms(50);
        }
for(i=0;i<5;i++)
        {
        LED2=1;
        delay_ms(50);
        LED2=0;
        delay_ms(50);
        }
}//fin du while(1)
```

8.5) ☺ ☺ BP influence nombre clignotements

```
char i, nb_cligno=2;        // Les lignes de déclaration sont toujours au début (juste après le « { » du main()
init();
while(1)
{
for(i=0;i<3;i++)
        {
        LED1=1;
        delay_ms(50);
        LED1=0;
        delay_ms(50);
        if(BP4==1)
                {
                nb_cligno=nb_cligno+1;
                delay_ms(100);
                break;
                }
        }
for(i=0;i<nb_cligno;i++)
        {
        LED2=1;
        delay_ms(50);
        LED2=0;
        delay_ms(50);
        if(BP4==1)
                {
                nb_cligno=nb_cligno+1;
                delay_ms(100);
                break;
                }
        }}//fin du while(1)
```

8.6) ☺ ☺ Séquence 3-2-3-2-3-2-5

```
char i,j;           // Les lignes de déclaration sont toujours au début (juste après le « { » du main()
init();
while(1)
```

```
{
for(i=0;i<3;i++)
        {
        for(j=0;j<3;j++)
                {
                LED1=1;
                delay_ms(50);
                LED1=0;
                delay_ms(50);
                }
        for(j=0;j<2;j++)
                {
                LED2=1;
                delay_ms(50);
                LED2=0;
                delay_ms(50);
                }
        }
for(i=0;i<5;i++)
        {
        LEDS=0b11111111;
        delay_ms(50);
        LEDS=0;
        delay_ms(50);
        }
}//fin du while(1)
```

Les exercices proposés sur l'algèbre de Boole ont été tirés de :
http://fr.wikiversity.org/wiki/Logique_de_base/Algèbre_de_Boole

Les corrections ainsi que des exercices supplémentaires s'y trouvent également.

10.1)

- a + a . b

Je factorise par rapport à a dans tous les termes:
= a (1 + b)
= a . 1
= a

a + a . b = a

10.2)

- a . (a + b)

Je développe a par rapport à la parenthèse :
= a.a + a.b
= a + a.b
Je factorise par rapport à a dans tous les termes: = a . (1+b)
= a . 1
= a

a . (a + b) = a

10.3)

- (a + b) . (a + $\overline{b}$)

Je développe la première parenthèse avec la deuxième :
= a.a + a.$\overline{b}$ + b.a + b.$\overline{b}$
= a + a.$\overline{b}$ + a.b + 0
= a + a.$\overline{b}$ + a.b
Je factorise par rapport à a dans tous les termes:
= a (1 + $\overline{b}$ + b)
= a . 1
= a

(a + b) . (a + $\overline{b}$) = a

10.4)

- (a + b) . (a + c)

Je développe la première parenthèse avec la deuxième :
= a.a + a.c + b.a + b.c
= a + a.c + a.b + b.c
Je factorise a dans tous les termes, sauf dans le dernier :
= a (1 + c + b) + b.c
= a.1 + b.c
= a + b.c

(a + b) . (a + c) = a + b.c

10.5)

$Q = (a+b+c) . (a+\overline{b}+c) . (a+\overline{b}+\overline{c})$

Je développe les deux premières parenthèses :

$Q = (a.a + a.\overline{b} + a.c + b.a + b.\overline{b} + b.c + c.a + c.\overline{b} + c.c) . (a+\overline{b}+\overline{c})$

$Q = (a + a.\overline{b} + a.c + a.b + 0 + b.c + a.c + \overline{b}.c + c) . (a+\overline{b}+\overline{c})$

$Q = (a + a.\overline{b} + a.c + a.b + b.c + \overline{b}.c + c) . (a+\overline{b}+\overline{c})$

Dans la première parenthèse, je factorise a autant que possible et dans le reste, je factorise c autant que possible :

$Q = [a . (1 + \overline{b} + c + b) + c . (b + \overline{b} + 1)] . (a+\overline{b}+\overline{c})$

$Q = (a.1 + c.1) . (a+\overline{b}+\overline{c})$

$Q = (a+c) . (a+\overline{b}+\overline{c})$

$Q = a.a + a.\overline{b} + a.\overline{c} + c.a + c.\overline{b} + c.\overline{c}$

$Q = a + a.\overline{b} + a.\overline{c} + a.c + \overline{b}.c + 0$

$Q = a + a.\overline{b} + a.\overline{c} + a.c + \overline{b}.c$

Je factorise a autant que possible $Q = a.(1 + \overline{b} + \overline{c} + c) + \overline{b}.c$

$Q = a.1 + \overline{b}.c$

$Q = a + \overline{b}.c$

$Q = a + \overline{b}.c$

11.1)

Après l'initialisation, on peut mettre dans un while(1) :

1ère façon

```
lcd_goto(0x00) ;
if ( (PORTA | 0b11111110) = = 0b11111110 )
        {lcd_puts("BP1 RELACHE") ;}
else
        {lcd_puts("BP1 ENFONCE") ;}
lcd_goto(0x40) ;
if ( (PORTA | 0b11111101) = = 0b11111101 )
        {lcd_puts("BP2 RELACHE") ;}
else
        {lcd_puts("BP2 ENFONCE") ;}
```

3ème façon

```
lcd_goto(0x00) ;
if ( (PORTA | 0b00000001) = = PORTA )
        {lcd_puts("BP1 ENFONCE") ;}
else
        {lcd_puts("BP1 RELACHE") ;}
lcd_goto(0x40) ;
if ( (PORTA | 0b00000010) = = PORTA )
        {lcd_puts("BP2 ENFONCE ") ;}
else
        {lcd_puts("BP2 RELACHE ") ;}
```

2ème façon

```
lcd_goto(0x00) ;
if ( (PORTA | 0b11111110) = = 0b11111111 )
        {lcd_puts("BP1 ENFONCE") ;}
else
        {lcd_puts("BP1 RELACHE") ;}
lcd_goto(0x40) ;
if ( (PORTA | 0b11111101) = = 0b11111111 )
        {lcd_puts("BP2 ENFONCE") ;}
else
        {lcd_puts("BP2 RELACHE") ;}
```

11.2)

Après l'initialisation, on peut mettre dans un while(1) :

1ère façon

```
lcd_goto(0x00) ;
if ( (PORTA & 0b00000100) = = 0b00000100 )
        {lcd_puts("BP3 ENFONCE") ;}
else
        {lcd_puts("BP3 RELACHE") ;}
lcd_goto(0x40) ;
if ( (PORTA & 0b00001000) = = 0b00001000 )
        {lcd_puts("BP4 ENFONCE") ;}
else
        {lcd_puts("BP4 RELACHE") ;}
```

3ème façon

```
lcd_goto(0x00) ;
if ( (PORTA & 0b11111011) = = PORTA )
        {lcd_puts("BP3 RELACHE ") ;}
else
        {lcd_puts("BP3 ENFONCE ") ;}
lcd_goto(0x40) ;
if ( (PORTA & 0b11110111) = = PORTA )
        {lcd_puts("BP4 RELACHE ") ;}
else
        {lcd_puts("BP4 ENFONCE ") ;}
```

2ème façon

```
lcd_goto(0x00) ;
if ( (PORTA & 0b00000100) = = 0b00000000 )
        {lcd_puts("BP3 RELACHE ") ;}
else
        {lcd_puts("BP3 ENFONCE ") ;}
lcd_goto(0x40) ;
if ( (PORTA & 0b00001000) = = 0b00000000 )
        {lcd_puts("BP4 RELACHE ") ;}
else
        {lcd_puts("BP4 ENFONCE ") ;}
```

11.3)

Après l'initialisation, on peut mettre dans un while(1) :

```
if (BP1= =1)
        {
        if ( (PORTB&0b10000000) = = 0b10000000 )        //si la LED1 est allumée
                {
                //on éteint la LED1
                PORTB = (PORTB & 0b01111111) ;
                }
        else    {
                //on allume la LED1
                PORTB = (PORTB | 0b10000000) ;
                }
        while(BP1 = = 1) ;          //on attend le relâchement de BP1
        }
if (BP2= =1)
        {
        if ( (PORTB&0b01000000) = = 0b01000000 )        //si la LED2 est allumée
                {
                //on éteint la LED2
                PORTB = (PORTB & 0b10111111) ;
                }
        else    {
                //on allume la LED2
                PORTB = (PORTB | 0b01000000) ;
                }
        while(BP2 = = 1) ;          //on attend le relâchement de BP2
        }
```

```
if (BP3= =1)
        {
        if ( (PORTB&0b00100000) = = 0b00100000 )        //si la LED3 est allumée
                {
                //on éteint la LED3
                PORTB = (PORTB & 0b11011111) ;
                }
        else    {
                //on allume la LED3
                PORTB = (PORTB | 0b00100000) ;
                }
        while(BP3 = = 1) ;          //on attend le relâchement de BP3
        }
if (BP4= =1)
        {
        if ( (PORTB&0b00010000) = = 0b00010000 )        //si la LED4 est allumée
                {
                //on éteint la LED4
                PORTB = (PORTB & 0b11101111) ;
                }
        else    {
                //on allume la LED4
                PORTB = (PORTB | 0b00010000) ;
                }
        while(BP4 = = 1) ;          //on attend le relâchement de BP4
        }
```

11.4)

On fait comme pour l'exercice 11.3) en remplaçant :

```
(BP1= =1)       PAR     ( (PORTA | 0b11111110) = = 0b11111111 )
(BP2= =1)       PAR     ( (PORTA | 0b11111101) = = 0b11111111 )        etc.
```

12.1) ☺ Fct 1 clignotement 5Hz

déclaration (dans prog.h) : `void cligno5Hz(void);`

implémentation (dans prog.c) :

```
void cligno5Hz()
        {
        LEDS=0b11111111;
        delay_ms(100);
        LEDS=0b00000000;
        delay_ms(100);
        }
```

boucle infinie :

```
while(1)
        {
        cligno5Hz();
        }
```

12.3) ☺ ☺ 2 appels Fct cligno puis pause

```
while(1)
        {
        cligno5Hz();
        cligno5Hz();
        delay_ms(1000) ;
        }
```

12.4) ☺ ☺ ☺ Nb clignos en paramètre (séquence 6-4-2)

déclaration (dans prog.h) : void cligno_x_fois(int nb);

implémentation (dans prog.c) :

```
void cligno_x_fois(nb)
        {
        int i;
        for(i=0;i<nb;i++)
                {
                LEDS=0b11111111;
                delay_ms(100);
                LEDS=0b00000000;
                delay_ms(100);
                }
        }
```

boucle infinie :

```
while(1)
        {
        cligno_x_fois(6);
        delay_ms(1000) ;
        cligno_x_fois(4);
        delay_ms(1000) ;
        cligno_x_fois(2);
        delay_ms(1000) ;
        }
```

13.1) ☺ ☺ Menu principal + 1 sous-menu

```
void main(void)
{
char menu_pr_en_cours=1 ;
init();
while(1)
{
if(menu_pr_en_cours==1)
        {
        lcd_goto(0x00);
        lcd_puts("MENU PRINCIPAL  ");
        lcd_goto(0x40);
        lcd_puts(" BP1 POUR CLIGNO");
        LEDS=0b11111111 ;
        if(BP1==1)
                menu_pr_en_cours=0 ;     //on pourrait aussi ajouter un délai dans la structure de choix
        }
else
        {
        lcd_goto(0x00);
        lcd_puts("SOUS-MENU   *RET");
        lcd_goto(0x40);
        lcd_puts("CLIGNO      *BP4");
        LEDS=0b10000000 ;
        delay_ms(100) ;
        LEDS=0b00000000 ;
        delay_ms(100) ;
        if(BP4==1)
                menu_pr_en_cours=1 ; //on pourrait aussi ajouter un délai dans la structure de choix

        }
}
}//fin du main
```

13.2) ☺ ☺ ☺ Menu principal + 1 sous-menu mais même BP

La résolution ressemble à l'exercice 13.1) mais ici il est indispensable d'ajouter un délai ou d'attendre le relâchement du bouton au sein de la structure de choix.
Il faut garder à l'esprit que lorsqu'on enfonce un bouton, la durée d'enfoncement sera au minimum de plusieurs dizaines de millisecondes, ce qui permet au PIC de faire des dizaines voire centaines de milliers d'instructions.

Correction de l'exercice récapitulatif, Chenillards

« prog.h »

```
void init(void);
void gere_menu_debut(void);
void affiche_menu_principal(void);
void affiche_menu_chenillard(void);
void affiche_menu_compte_bp(void);
char gestion_chenillard(void);
char gestion_menu_compte_bp(char);
char affiche_menu_indique_nb_bp(void);
```

« prog.c »

```
#include<pic18.h>
#include<stdio.h>
#include"prog.h"
#include"lcd.h"
#include"delay.h"


#define  BP1 RA0
#define  BP2 RA1
#define  BP3 RA2
#define  BP4 RA3

#define  LED1 RB0

#define  menu_debut              1
#define  menu_principal          2
#define  menu_chenillard         3
#define  menu_compte_bp          4
#define  menu_indique_nb_bp      5

char ligne1[16];

void main(void)
{
char menu_en_cours=menu_debut;
char nb;
init();
```

```
while(1)
{
	switch(menu_en_cours)
		{
		case menu_debut :

			gere_menu_debut();
			menu_en_cours=menu_principal;
			break;

		case menu_principal :

			affiche_menu_principal();
			if(BP1)
				menu_en_cours=menu_chenillard;
			else if(BP2)
				menu_en_cours=menu_compte_bp;
			else if(BP3)
				menu_en_cours=menu_indique_nb_bp;
			else if(BP4)
				menu_en_cours=menu_debut;
			break;

		case menu_chenillard :

			affiche_menu_chenillard();
			while(gestion_chenillard());
			while(BP4);	//attend le relachement
			menu_en_cours=menu_principal;
			break;

		case menu_compte_bp :
			nb=0;
			while(gestion_menu_compte_bp(nb))
				{
				if (BP1)
					{
					while(BP1);	//attend le relachement
					nb++;
					}
				}
			while(BP4);	//attend le relachement
			menu_en_cours=menu_principal;
			break;

		case menu_indique_nb_bp :
			while (affiche_menu_indique_nb_bp());
			while(BP4);	//attend le relachement
			menu_en_cours=menu_principal;
			break;

		}
}//fin du while(1)

}//fin du main
```

```
void init()
        {
        TRISA=0b11101111;
        ADCON1=0x0f;
        PORTA=0;
        TRISB=0b00000000;
        PORTB=0;
        TRISC=0b00000000;
        PORTC=0;
        lcd_init();
        lcd_clear();
        PORTB=0;
        }

void gere_menu_debut()
        {
        char i=0;
        lcd_goto(0x00);
        lcd_puts("EXAMEN D'INFO A.");
        lcd_goto(0x40);
        lcd_puts("PATIENTEZ 5s    ");
        for(i=0;i<5;i++)
                {
                LED1=1;
                delay_ms(500);
                LED1=0;
                delay_ms(500);
                }
        }

void affiche_menu_principal()
        {
        lcd_goto(0x00);
        lcd_puts("CHE*CPT*NBR*RET.");
        lcd_goto(0x40);
        lcd_puts("BP1*BP2*BP3*BP4 ");
        }

void affiche_menu_chenillard()
        {
        lcd_goto(0x00);
        lcd_puts("SOUS-MENU 1*RET.");
        lcd_goto(0x40);
        lcd_puts("CHENILLARD *BP4");
        }

char gestion_chenillard()
        {
        char i=0;
        char doit_continuer=1;
        if (BP4)
                doit_continuer=0;
        else
                {
                delay_ms(200);
                PORTB=1;
                for (i=0;i<7;i++)
                        {
                        if (BP4)
                                {
```

```
                                        doit_continuer=0;
                                        break;
                                        }
                                delay_ms(200);
                                PORTB=PORTB<<1;
                                }
                        }
                return doit_continuer;
                }

char gestion_menu_compte_bp(char nb)
                {
                char espace_2_char[2];
                char nb_enfoncements=nb;
                char doit_continuer=1;
                if (BP4)
                        doit_continuer=0;
                else
                        {
                        lcd_goto(0x00);
                        lcd_puts("SOUS-MENU 2*RET.");
                        lcd_goto(0x40);
                        lcd_puts("BP1 ");
                        sprintf(espace_2_char,"%2d",nb_enfoncements);
                        lcd_puts(espace_2_char);
                        lcd_puts(" FOIS*BP4");
                        }
                return doit_continuer;
                }

char affiche_menu_indique_nb_bp()
                {
                char espace_1_char[1];
                char nb_boutons_enfonces=BP1+BP2+BP3;
                char doit_continuer=1;
                if (BP4)
                        doit_continuer=0;
                else
                        {
                        lcd_goto(0x00);
                        lcd_puts("SOUS-MENU 3*RET.");
                        lcd_goto(0x40);
                        sprintf(espace_1_char,"%1d",nb_boutons_enfonces);
                        lcd_puts(espace_1_char);
                        lcd_puts(" BP ENF.  *BP4 ");
                        }
                return doit_continuer;
                }
```