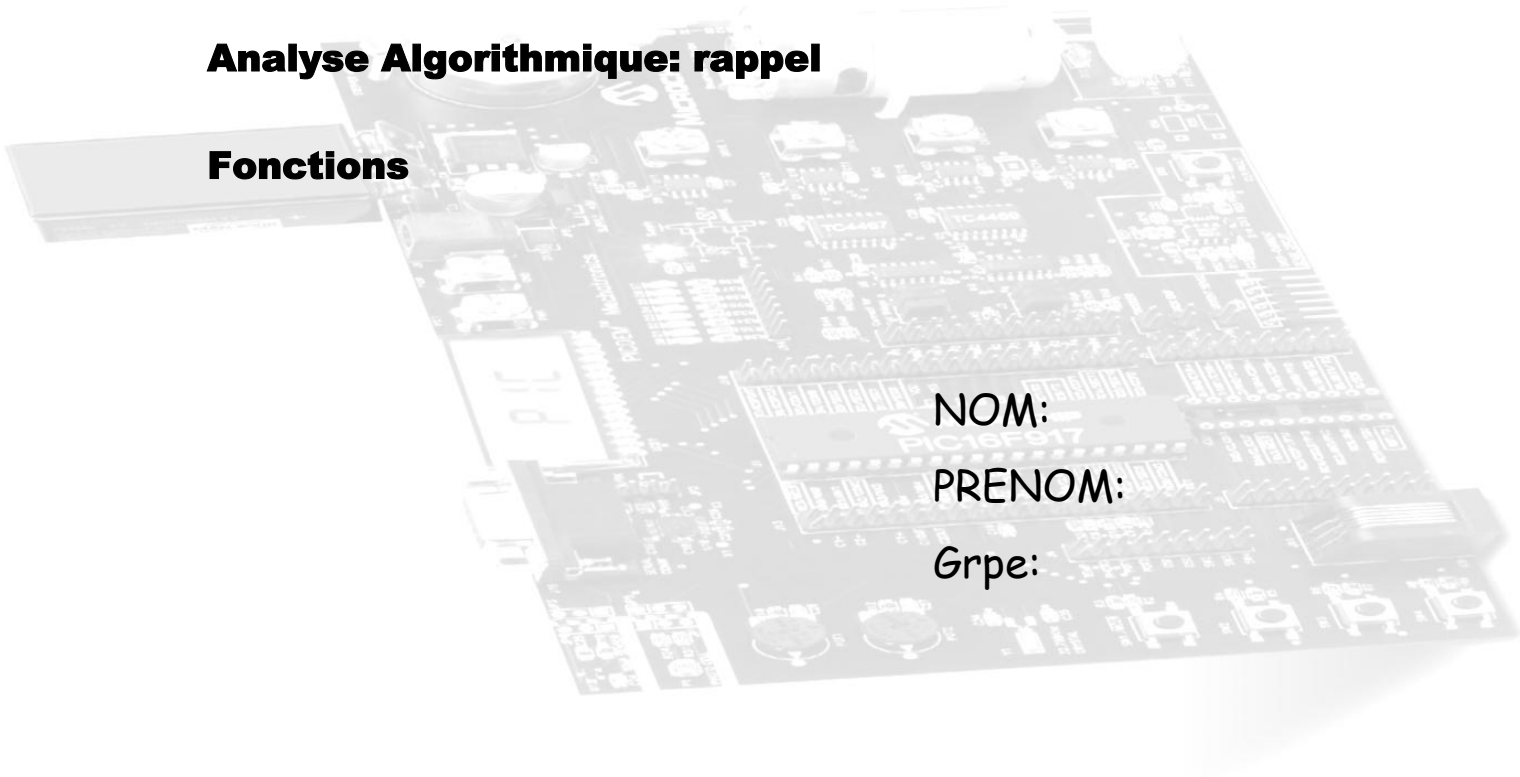
PIC

Analyse Algorithmique: rappel

Fonctions



NOM:

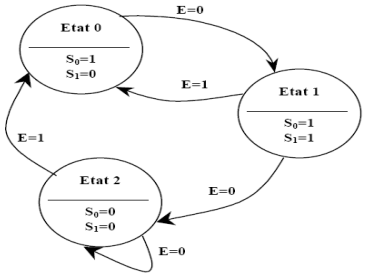
PRENOM:

Grpe:

Méthodologie

Éléments de méthodologie

- L'écriture d'un programme (surtout si c'est un programme gérant une application industrielle complexe) doit débuter par une ANALYSE.
- Cette analyse permet de découper le programme opérationnel en un certain nombres de fonctions élémentaires plus simples à mettre au point.
  - ✓ Exemple : Initialisations, Entées/ Sorties d'informations, Traitement, Etc …
- Pratiquement : Dans un programme d'application industrielle le Programme Principal est relativement succinct : il fait appel à un grand nombre de FONCTIONS (ou sous-programmes) spécialisées.
- Marche à suivre pour effectuer l'analyse :
  - ✓ Lecture du problème.
  - ✓ Définition des différentes actions à réaliser
  - ✓ Chronologie des évènements et des actions à réaliser.
  - ✓ Ecriture des suites d'opérations à réaliser (écriture de l'ALGORITHME)



L'algorithme

- Il se compose d'une suite d'instructions qui sont exécutées de façon séquentielle les unes après les autres sauf s'il y a rupture de séquence lors d'un **branchement conditionnel** ou de **l'appel d'une fonction**
- Des exemples

```
y ← 3
val ← entree
Si (entree  <y+2)
   val ← 0xff
       sinon
    val ← 0xaa
Fin Si
y ← 5
a ←0
```

```
TantQue  (Température < 20 degrés)
        •
        •  Activer le Chauffage
        •
Fin TantQue
```

Algorithme

L'affectation ←

- Elle permet de donner une valeur à une variable. Affecter x à y, signifie que y prend la valeur de x
- *L'opération d'affectation possède un sens et sera notée* ←
  - ✓ variable ←expression

exemple:

```
x ←x+2
y ←x
a ←0
```

Les Boucles

La boucle TANTQUE (Condition) … FINTANTQUE

- ✓ Cette instruction permet de répéter l'exécution d'une instruction tant que l'expression (entre parenthèse) est vraie (en C, différent de 0).
- ✓ On teste d'abord la condition

Syntaxe :

```
TANTQUE (Condition Vraie)
|     instruction(s)
FINTANTQUE
```

Exemple:

Analyse Algorithmique:

```
entier : i,x
i ←0
x ←5
TANTQUE ( i<9 )
| x ←x+3
| i ←i+2
FINTANTQUE
```

Programme en C :

```
int i,x ;
i=0;
x=5;
while (i<9)
        {
        x=x+3 ;
        i=i+2;
        }
```

La boucle FAIRE …… TANTQUE (Condition Vraie)

- ✓ Cette instruction permet également de répéter l'exécution d'une instruction tant que l'expression (entre parenthèse) est vraie (en C, différent de 0).

Syntaxe :

```
FAIRE
|       instruction(s)
TANTQUE (Condition Vraie)
FIN FAIRE TANTQUE
```

Exemple1 :

Analyse littérale :

```
Faire
|   Désactiver le Chauffage
TantQue (Température > 20 degrés)
Fin FaireTantQue
```

Algorithme

Les Choix et les Tests

Le choix SI… FINSI

✓ Cette instruction permet d'exécuter une instruction (ou un bloc d'instructions) **si une expression est évaluée comme vraie**

Syntaxe :

```
SI (expression)
        instruction(s)
FINSI
```

```
SI (x == 8)
        x ←-x
        y ←y+9
FINSI
```

Le choix SI… SINON… FINSI

Syntaxe :

```
SI (expression)
        instruction(s)
  SINON
        instruction(s)
FINSI
```

Exemple :

```
SI (Température >20°)
        Couper le chauffage
  SINON
        Activer le chauffage
FINSI
```

Le choix multiple

syntaxe :

```
selon que la (variable de contrôle)
{
  est égale à cte1 :   …….
                       exécuter la séquence1 d'instruction(s);
                       ……
                       break;
  est égale à cte2 :   …….
                       exécuter la séquence2 d'instruction(s);
                       ……
                       break;
  ou par défaut:       exécuter une autre séquence;
}
```

Codage en C exemple

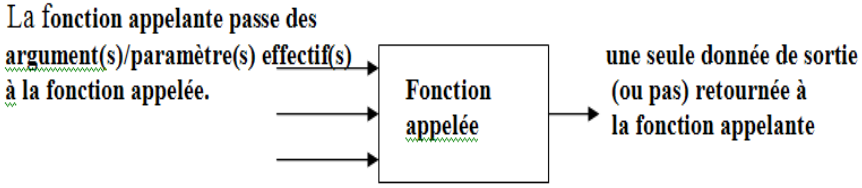
```
switch (n)
{
 case 1 :  a = 40 ;
           b = 0X60 ; break ;// si n = 1 exécuter les 2 instructions et sortir
case 2 :   a = 2 ;
           c = 65 ; break ; // si n = 2 exécuter les 2 instructions et sortir */
case 'A' : d  = 4 ; break ;/* si n = au code ASCII de A soit 0X41 …….*/
default : var = 6 ;/* si n  ne répond pas au 3 premiers cas alors exécuter var = 6
}
```

Fonctions

Les Fonctions

- Les fonctions permettent de découper un programme en plusieurs parties spécialisées.
- Elles sont particulièrement intéressantes si elles sont appelées de façon répétitive.
- Le main ( ) n'est qu'une fonction qui joue un rôle privilégié. Cette fonction est exécutée en premier juste après l'initialisation des variables globales

Principe d'une fonction



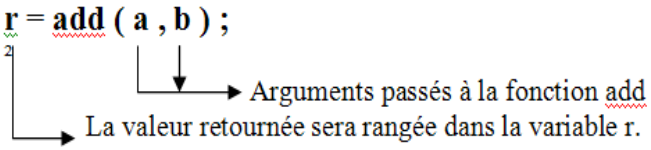
Le passage des paramètres effectifs vers les paramètres formels se fait dans l'ordre de leur écriture dans le programme appelant

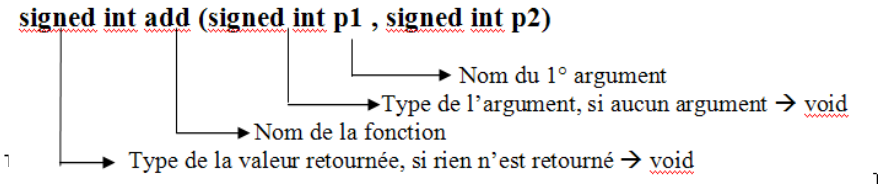
Un exemple

```
void main (void)
{
a = 10 ;
b = 120 ;
 /* appel de la fonction pour
exécution, a et b paramètres effectifs
passés à la fonction, la variable
retournée ira dans la variable  r */

r =  add (a, b) ;
}
```

```
// LA FONCTION APPELéE
signed int  add  (signed int p1, signed int p2)
/* p1 et p2 arguments (paramètres formels) */
{
signed int p3 ;              /* déclaration d'une variable locale */
p3 = p1 + p2 ;
return  p3 ;
/* le contenu de la variable p3 est retourné dans la variable r */
}
```





Fonctions

Transmission d'une adresse

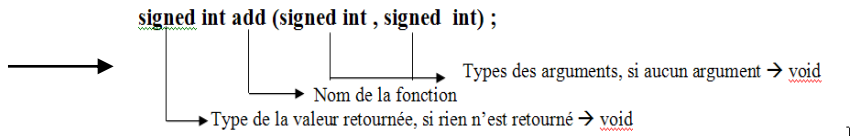
- plutôt que de transmettre le contenu d'une variable copie on transmet l'adresse où est stockée la variable
- C'est d'un moyen indirect de modifier le contenu de cette variable directement dans le corps de la fonction

```
/* Déclaration du prototype de la fonction */
/*  signed int* indique que l'on passe l'adresse d'une variable de type signed int */
void calcul (signed int*) ;
unsigned char a ;/* déclaration de la variable a */
void  main  (void){
a=3 ;
/*  appel de la fonction calcul à laquelle on passe l'adresse de la  variable a ( &a représente l'adresse de la variable a)  */
calcul  (&a) ;}
```

```
/* Ecriture de la fonction */
void calcul (signed int *adn)
/*  le  pointeur d'adresse  adn  prend la valeur de l'adresse de la  variable a   */
{
*adn = 2 ;        /*   *adn représente le contenu de la variable de type signed int ayant pour adresse adn  qui est aussi l'adresse de la variable a    */
}
```

Apprendre à utiliser des fonctions existantes

- Un indicateur précieux: le prototype de la fonction
  - ✓ Constitue la déclaration de la fonction.



- Application: utilisation de la librairie MPLAB C18
  - ✓ L'environnement de développement fournit aux programmeurs un ensemble de fonctions toutes faites pré compilée et distribuée sous forme de librairie. Cette librairie est liée à votre projet par l'intermédiaire du LINKER script au moment de l'édition de liens.
  - ✓ Elle facilite la tâche du concepteur en fournissant des fonctions de haut niveau permettant d'accéder et de manipuler de manière aisée les périphériques du PIC.

Delay1TCY

Function:	Delay 1 instruction cycle (TCY).
Include:	`delays.h`
Prototype:	`void Delay1TCY( void );`
Remarks:	This function is actually a `#define` for the NOP instruction. When encountered in the source code, the compiler simply inserts a NOP.
File Name:	`#define in delays.h`

Delay100TCYx

Function:	Delay in multiples of 100 instruction cycles (TCY).
Include:	`delays.h`
Prototype:	`void Delay100TCYx( unsigned char unit );`
Arguments:	***unit*** The value of ***unit*** can be any 8-bit value. A value in the range [1,255] will delay (***unit*** * 100) cycles. A value of 0 causes a delay of 25,600 cycles.

Fonctions

- **Un exemple: utilisation de l'écran LCD**
  - CARTE VERTE PICDEM: Recopier les fichiers xlcd200V.h, xlcd200V.lib et 18f4520ixlcd200V.lkr dans le répertoire de votre projet. (...R pour carte rouge)
  - Ajouter ces fichiers à votre projet.
  - Inclure le fichier xlcd200V.h dans chaque fichier source utilisant une fonction lcd.

```
// à compiler avec l'option memory
//model large pour ne //pas avoir
//de warning sur printf
#include <p18cxxx.h>

// configuration PICDEM2+ quartz
#pragma config OSC = HS
#pragma config WDT = OFF
#pragma config LVP = OFF
#pragma config PBADEN = OFF


#include <stdio.h>

#include "xlcd200V.h"

#define S2 LATAbits.LATA4

unsigned char
TacheTestAppuiTouche_S2();
unsigned char touche=0;
unsigned int compteur=0;
```

```
void main()
{
        unsigned char toucheFM;
        TRISAbits.TRISA4=1;// S2 en entrée
        TRISB=TRISB&0xF0; // RB4:0 en sortie
        LATB=0; //éteint les LEDs

        // initialise le lcd  écran effacé curseur pos 0
        OpenXLCD(OPEN_PICDEM_LCD);

        // redirige sortie standard sur le lcd
        stdout = _H_USER ;
        // message initial sur LCD
        gotoXLCD(LCD_LINE_ONE);
        puts("compteur appui touche":");
        gotoXLCD(LCD_LINE_TWO);
        printf("%d",compteur);
        LATB=compteur; // sortie aussi sur les LEDs

        for(;;)
                {
                toucheFM=TacheTestAppuiTouche_S2();
                if (toucheFM=1)
                  {
                    gotoXLCD(LCD_LINE_TWO);
                    printf("%d",compteur);
                // sortie aussi sur les LEDs
                    LATB=compteur;
                }
}
```

```
// renvoie 1 si detection d'un front montant sur la
//touche
unsigned char TacheTestAppuiTouche_S2()
{
 static unsigned char etat=0;

switch (etat)
  {
    case 0 : touche=0;
                    if(S2==0)
           {etat=1}; // si touche appuyee alors etat1
           break;
    case 1 : touche=1; // active etat touche
           etat=2;  // va au prochain coup à l'état 2
           break;
   case 2 : touche=0;  // désactive etat touche
                  if(S2==1)
          {etat=0;}; // si bouton relaché alors va etat 0
          break;
   default: etat=0;
  }
return(touche); // renvoye etat touche
}
```