Langage C

Cette présentation du langage C contient tout ce qui est nécessaire pour créer et comprendre la partie logicielle de programme simple. Ce n'est pas un cours exhaustif et de nombreux aspects du langage C (tableaux, structures, pointeurs,…) ont été volontairement omis afin de ne pas alourdir cette présentation.

1 Les bases du langage C

Pourquoi un tel succès? C est un langage:

- ***Universel ➔ Vaste domaine d'applications***
- **Compact ➔ Instructions simples et efficaces**
- **Moderne ➔ structuré, déclaratif**
- **Près de la machine ➔ rapidité, programmation des microcontrôleurs**
- **Portable ➔ Un grand choix de compilateurs C le rend indépendant de la machine et du système d'exploitation**
- **Extensible ➔ Nombreuses bibliothèques disponibles selon l'application**

Mais attention!

- **Utilisation de fonctions préfabriquées ➔ limitation de la portabilité**
- **Format libre + code compact ➔ programme illisible**
- **Beaucoup de liberté laissée au programmeur ➔ … et des responsabilités**

2 Exemple de programme en langage C

```
#include <Servo.h>              // Directives de précompilation
#define  valcli 0x90

Int  cmpt;                      // zone de déclaration des variables globales

void main()                     // programme principal
{
        char i ;                        // variables locales
        cmpt = 8 ;                      // instructions

        while (true)
        {
                for ( i = 0 , i <= cmpt , i ++ )
                {
                        digitalWrite(led, HIGH);
                        delay(valcli);
                        digitalWrite(led, LOW) ;
                        delay(valcli);
                }
        delay(2000);
        }
}
```

3 Généralités

- **Structure d'un programme**

  Un programme en C est composé essentiellement de fonctions et de variables.
  La fonction main( ) est le programme principal. Il est délimité par des accolades "{" et "}".
  Le programme C minimal est : `main ( ) { }`

- **Commentaires**

  Commentaire sur une seule ligne : double slash (//) suivi de votre commentaire.
  Commentaire sur plusieurs lignes : pour indiquer le début du commentaire : tapez un slash suivi d'une étoile (/) ; pour indiquer la fin du commentaire : tapez une étoile suivie d'un slash (*/).*

- **Instructions**

  Chaque instruction est terminée par un point virgule (;).
  Il se peut que plusieurs instructions soient regroupées pour constituer une instruction composée; dans ce cas elle est délimitée par un couple d'accolades "{" …… "}" et forme un bloc.

4 Les types simples

En C, les variables doivent être déclarées avant d'être manipulées par des opérateurs.
La déclaration d'une variable consiste à définir son nom et son type.

- **Variable binaire (valeur 0 ou 1)**

```
boolean  ready ;          // déclaration d'une  variable de type binaire nommée " ready "
ready = 0 ;               // affectation de la valeur 0 à la variable ready
ready = read(BP) ;        // ready prend la valeur renvoyée par la fonction read()
```

- **Entier 8 bits non signé (valeur de 0 à 255)**

```
byte  c ;                 // déclaration de la variable 'c' de type  entier 8 bits non signé
c = 12 ;                  // affectation d'une  valeur  décimale à la variable c
c = 0x0C ;                // affectation d'une valeur  hexadécimale
```

- **Entier 16 bits signé (en complément à 2) valeur de –32768 à 32767 (-2^{15} à $2^{15} – 1$)**

```
int li ;                  // déclaration de la variable 'li'  (entier 16 bits signé)
li = 0xFFF0 ;             //  en décimal ? :  - ……
```

- **Entier 16 bits non signé valeur de 0 à 65535 (0 à $2^{16} – 1$)**

```
unsigned int cmpt ;
cmpt =  1880 ;            // affectation de la valeur décimale 1880 à la variable cmpt
cmpt = 0x0758;            // même opération en hexadécimal
```

- **Virgule flotante 32 bits (signé) mantisse 23 bits , exposant 8 bits et 1 bit de signe**
  Valeur de -3.4028235E+38 à +3.4028235E+38

```
float  gnb ;
gnb =  1254.89652
gnb = 0x0758;             // même opération en hexadécimal
```

- **Caractère ou entier 8 bits signé en complément à 2 (valeur de -128 à 127)**

```
signed char  i = 18 ;     // déclaration de la variable 'i' puis affectation de la valeur 18 à i
signed char  sc, varx ;   // déclaration de deux variables (sc et varx)
i = -34 ;
i = 0b11011110 ;          // en  binaire
i= 'a'  ;                 // affectation du code ASCII de 'a' (Ox61) à la variable 'i'
```

5 Les opérateurs

- **Affectation**

  = **Affectation simple**

  <nom_variable> = <expression> ; **// range <expression> en <nom_variable>**

- **Calcul**

  + **addition**

  - **soustraction**

  ***** **multiplication**

  / **division entière**

  % **reste de la division entière (prononcer « modulo »)**

  Exemples

  ```
  a = 15 ; b =  10 ;    // ranger 15 dans a et 10 dans b
  c = a + b ;           // c = 25
  d = a – b ;           // d =
  e = 3 * c ;           // e =
  x = a / b ;           // x =
  y = a % b ;           // y =
  ```

- **Incrémentation**

  i = i + 1 ; peut s'écrire de manière plus condensée

  ++i **pré-incrémentation**

  i++ **Post-incrémentation**

  a = n++ ; est équivalent à :

  a = n ;

  n = n + 1 ;

- **Décrémentation**

  De la même manière l'opérateur de décrémentation s'écrit --

  Exemples

  ```
  a = 5 ; n = 6 ; i = 0 ;
  i++ ;          // résultat : i =
  c = a++ ;      // résultat : c =    a =
  d = ++c ;      // résultat : d =    c =
  x = --n ;      // résultat : x =    n =
  y = x-- ;      // résultat : y =    x =
  ```

- **logiques**

  && **et logique (and)**

  || **ou logique (or)**

  ! **Négation logique (not)**

- **De comparaisons**

==	**égal à**	**!=**	**différent de**
>	**supérieur à**	**>=**	**supérieur ou égal à**
<	**inférieur à**	**<=**	**inférieur ou égal à**

➔ Les opérateurs logiques et de comparaisons renvoient les valeurs VRAI ou FAUX

➔ En C, FAUX est la valeur 0 , VRAI est tout le reste

Exemples

```
a = 4 ; b = 6  ; c = 6 ;
 a > b ;                          // résultat : FAUX
 b == c ;                         // résultat :
 ( a == 4 ) && ( b != 2 ) ;       // résultat :
 ( a == 5 ) || ( b >= 6 ) ;       // résultat :
 !( a == 4 ) ;                    // résultat :
```

- **Manipulation de bits**

~ complément à 1
& ET bit à bit
| OU bit à bit
^ OU EXCLUSIF bit à bit
>> n décalage à droite de n bits
<< n décalage à gauche de n bits

Exemples

```
a = 13 ; b = 6 ;          // a = 0b00001101   b= 0b00000110
c = ~a ;                  // c = 0b11110010 = 0xF2
d = a & b ;               // d =
e = a | b ;               // e =
f = a ^ 0x0F              // f =
g = a >> 2                // g =
h = b << 3                // h =
```

6 Les instructions de controle :

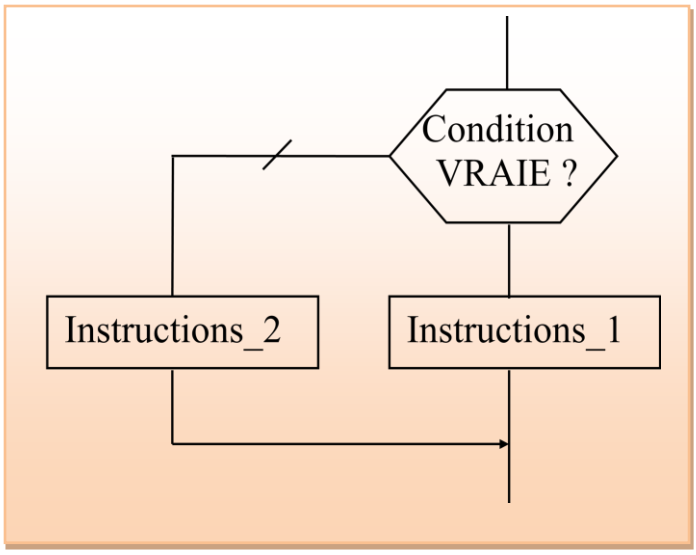
- **La structure alternative : « IF … ELSE »**

⇨ **Si… alors … sinon …**

```
if (condition)
{
        instructions_1     //  si VRAI
}
else              // facultatif
{
        instructions_2     //  si FAUX
}
```

Exemple :

```
a = 5 ; b = 7 ;
if ( a > b )
        max = a ;
else
        max = b ;
// résultat : max =
```



- **La structure alternative : « switch… case »**

Exemple :

```
switch (a) {
 case 1:
    b=a-1 ;
    break;
case 2:
    b=a-2 ;
    break;
default:
    b = 0 ;
}                                 //  résultat : b=
```

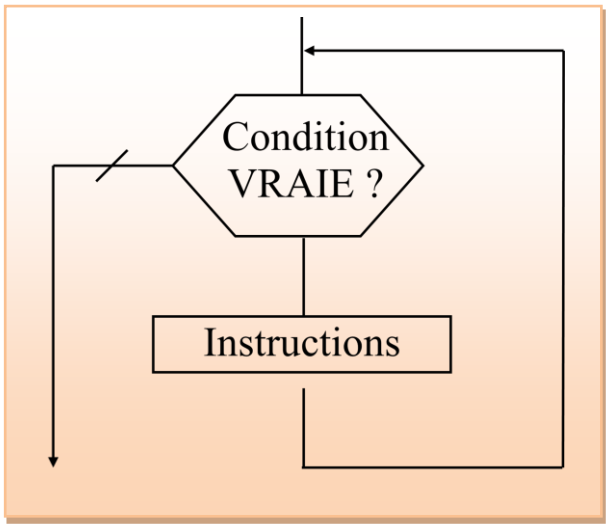
- **La structure répétitive « while »**

⇨ **tant que … faire …**

```
while (condition)
{
        instructions
}
```

Exemple

```
        a = 0b11000000 ;
        i = 0 ;
        while ( i < 3 ) {
                a = a >> 2 ;
                i + + ;
        }
        // i prend les valeurs successives :…
        // la boucle est effectuée … fois
        // Résultat : a = …
```



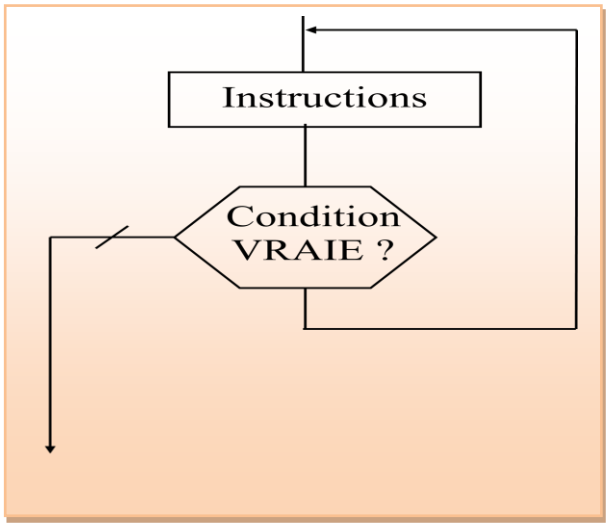
- **La structure répétitive « do … while »**

⇨ **faire … tant que …**

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

Exemple

```
        a = 4 ;
        i = 3 ;
        do      {
                a = a * a ;
                i-- ;
        } while( i > 0 )
        // i prend les valeurs successives :…
        // la boucle est effectuée … fois
        // Résultat : a = …
```



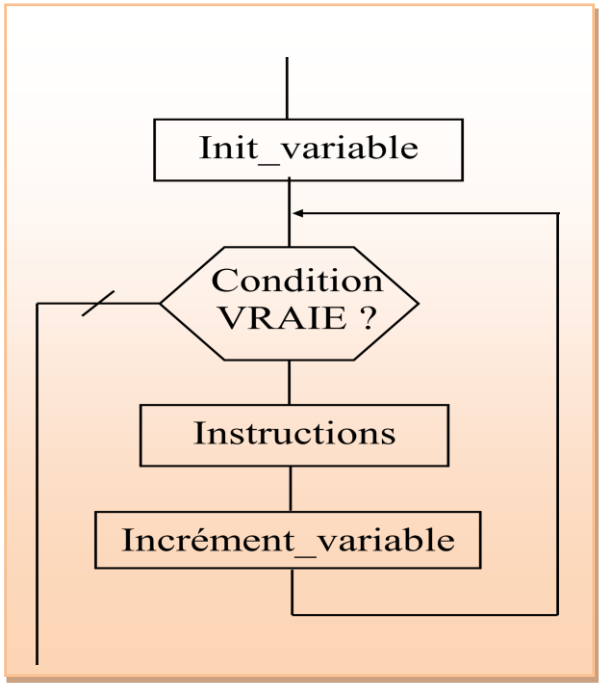
- **La structure répétitive « for »**

⇨ **pour … allant de … à … faire …**

```
for( init_variable , condition , incrément_variable )
{
        instructions
}
```

Exemple

```
        tot = 0 ;
        for( n = 1 ; n < 5 ; n ++ )
                tot = tot + n ;
        // n prend les valeurs successives : …
        // n = … interrompt la boucle
        // la boucle est effectuée … fois
        // Résultat : tot = …
```



7 Les fonctions

- **Introduction**

  Une fonction est un morceau de programme réalisant une tâche bien définie.
  Elle peut être appelée par le programme principal ou par d'autres fonctions
  Elle facilite l'écriture de gros programmes (programmation modulaire)

- **Définition d'une fonction**

```
Type_valeur_retournée  Nom_fonction (Type  Nom_paramètres)
{
        instructions
 }
```

  - ➔ **La valeur retournée par la fonction est précédée du mot clé *« return »***
  - ➔ **Si la fonction ne retourne pas de valeur ou n'utilise aucun paramètres on utilise le mot clé « *void* »**
  - ➔ **Avant d'être définie, la fonction doit avoir été déclarée**

- **Création et utilisation d'une fonction**

```
int carre(int) ; // déclaration (ou  prototype ) de la fonction carre()
void main ( void )      {
        int i , x ;
        for ( i = 1 ; i < 10  ; i++ )
        x = carre( i ) ;  // appel de la fonction carre()
}
int carre ( int n )        // définition de la fonction carre()
{
        return n * n ;  //  valeur retournée par la fonction
}
        // résultats : x = 1, 4, …
```

- **Variables globales et variables locales**

Une variable est **globale** lorsqu'elle est déclarée à l'extérieur des fonctions.
Elle est alors connue par toutes les fonctions du programme.
Si la variable est définie à l'intérieur d'une fonction elle est dite **locale**.
Elle n'est alors connue qu'à l'intérieur de la fonction où elle est déclarée.

```
Exemple
char n ;                  // variables globales
int i ;
void main( void )
{
        bit x ;  // variable locale à la fonction main( )
        ……
}
fct1( )
{
        char p ;          // variable locale à la fonction fct1( )
        ……..
}
```

8 Le Préprocesseur

Avant la compilation, un programme appelé **préprocesseur** transforme le fichier source à partir d'un certains nombres de directives.
Ces directives sont toujours introduites par par un mot commençant par le symbole #

- **La directive #INCLUDE**
  Elle permet d'incorporer, avant compilation, le texte figurant dans un fichier quelconque.

Par exemple,

```
#include "test.h "
```

demande d'incorporer le fichier en-tête test.h

- **La directive #DEFINE**
  Elle permet de définir des symboles. Par exemple,

```
#define led PIN_B3
```

demande de substituer au symbole "led "le texte "PIN_B3 "chaque fois que ce symbole apparaîtra dans la suite du programme source.

Bibliographie

Site sur le net : https://openclassrooms.com/fr/courses/19980-apprenez-a-programmer-en-c/13300-vous-avez-dit-programmer