Leçon N°5

AFFICHAGE DES DONNEES

I- INTRODUCTION

La conception des systèmes à microcontrôleurs implique parfois l'affichage des données à l'utilisateur. A cet effet, on peut utiliser des afficheurs 7 segments, l'afficheur de caractère à cristaux liquides LCD et l'afficheur LCD à écran graphique. Dans ce chapitre, on étudie et on illustre ces dispositifs par des exemples.

II- AFFICHAGE 7 SEGMENTS

Un afficheur 7 segments est un dispositif qui permet de visualiser un nombre limité de caractères essentiellement numériques, mais il est possible de visualiser quelques caractères comme : A, B, C, D, E ou F. L'afficheur 7 segments est un ensemble de LED, disposées de sorte qu'on visualise les caractères en activant les segments convenables. A chaque segment, on attribue une lettre de "a" jusqu'à "g". La description et l'apparence physique de ces afficheurs sont illustrées sur les images suivantes :

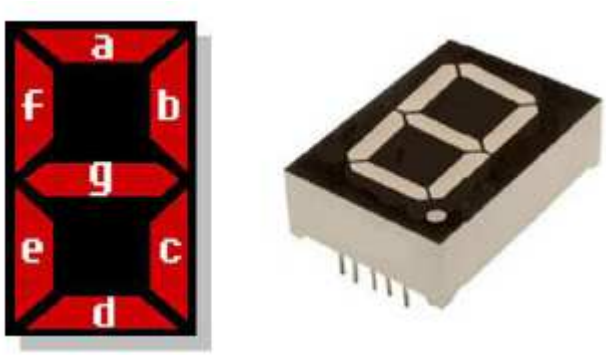

Figure 1

Les afficheurs 7 segments sont fabriqués en deux formats ; anode commune et cathode commune. Ils existent également dans un format dynamique. Ces derniers utilisent deux chiffres ou plus dans un seul boitier reliant tous les segments en parallèle, mais avec des bornes communes séparées. Les figures suivantes montrent des afficheurs 7 segments dans leur forme dynamique :

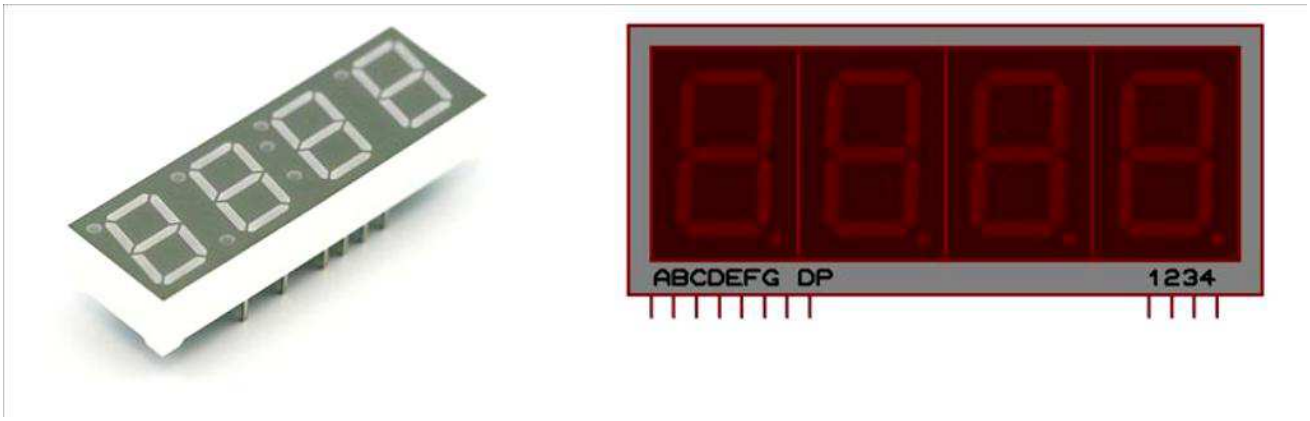

Figure 2

L'entrée DP, qu'on peut voir sur la figure ci-dessus, est le huitième segment. Il est mis en œuvre dans certains affichages et correspond à la virgule, il est utilisé si la demande l'exige.

1. Contrôle de l'affichage à 7 segments

Pour effectuer le premier exemple avec cet afficheur, on doit savoir comment réaliser l'affectation des broches du PIC. A chaque segment de l'afficheur, on doit relier une broche du PIC. Pour faire une conception optimale, on peut attribuer les broches en ordre consécutif d'un port, par exemple le segment "a" est relié à la broche RB0, le "b" à la broche RB1, et ainsi de suite, jusqu'au segment "g" à la broche RB6. Cependant, le développeur peut faire l'affectation des broches d'une manière arbitraire.

Il est important de connaitre un outil contenu dans le logiciel MikroC PRO qui permet d'éditer les digits de l'afficheur. Pour cela, on utilise *Tools* dans la barre de menu, et on choisit l'outil *Seven Segment Editor.*

Avec cette action, ressort une nouvelle fenêtre où on peut éditer d'une manière simple les segments de l'afficheur à 7 segments.

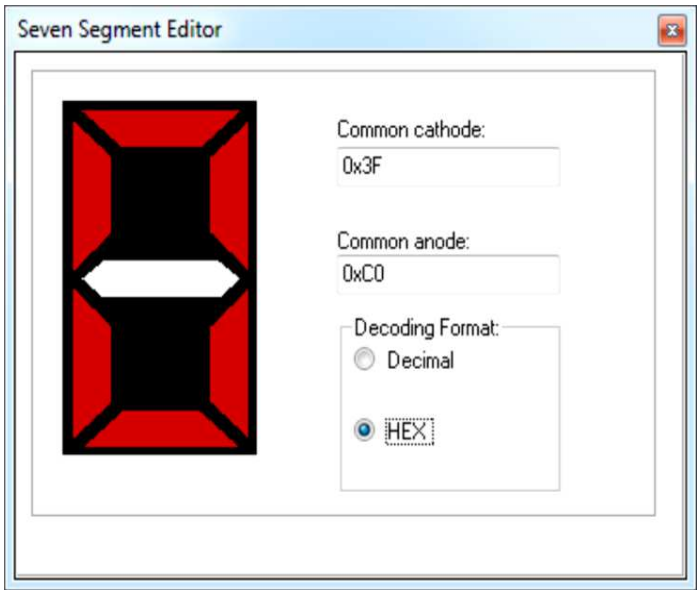

Figure 3

L'utilisation de cet outil signifie que toutes les broches de l'afficheur sont attribuées au même port, et dans un ordre consécutif comme on a vu ci-dessus. Pour la visualisation des chiffres dans l'afficheur, on doit organiser les informations de ces digits dans un ordre consécutif, dans le cas de cet exemple de 0 à 9. A cette fin, la forme la plus simple est l'utilisation d'un tableau de données contenant les codes des 10 chiffres. Dans l'exemple suivant, on déclare un tableau constant avec les codes des 10 chiffres, et on utilise un afficheur à cathode commune.

```
const unsigned short DIGITS[] =
{
        0x3F, //Code du digit 0
        0x06, // Code du digit 1
        0x5B, // Code du digit 2
```

```
        0x4F, // Code du digit 3
        0x66, // Code du digit 4
        0x6D, // Code du digit 5
        0x7D, // Code du digit 6
        0x07, // Code du digit 7
        0x7F, // Code du digit 8
        0x6F, // Code du digit 9
};
```

Pour afficher les chiffres dans l'afficheur à 7 segments, cet exemple va utiliser le port B du PIC 16F84A. La visualisation des numéros est contrôlée par une routine, qui change le chiffre après une temporisation réalisée par une fonction *delay_ms*, cette fonction qui est prédéfinie dans les bibliothèques du compilateur. Cette fonction a pour paramètre d'entrée un entier représentant le temps en millisecondes, pendant lequel le PIC effectue un retard, de la même manière on peut utiliser la fonction *delay_us*, qui est identique à *delay_ms* mais en microsecondes.

Alors, le code source du PIC pour cet exemple est le suivant :

```
const unsigned short DIGITS[] =
{
        0x3F, //Code du digit 0
        0x06, //Code du digit 1
        0x5B, //Code du digit 2
        0x4F, //Code du digit 3
        0x66, //Code du digit 4
        0x6D, //Code du digit 5
        0x7D, //Code du digit 6
        0x07, //Code du digit 7
        0x7F, //Code du digit 8
        0x6F, //Code du digit 9
};
void main ( void )
{
        unsigned short COMPTEUR=0;
        TRISB = 0; // Configuration du port B en sortie
        while (1)    //Boucle infinie
        {
```

```
                PORTB = DIGITS[COMPTEUR]; //Visualisation du chiffre correspondant
                                          // à la valeur de la variable COMPTEUR.
                COMPTEUR++; //Incrémentation la valeur de compteur.
                delay_ms(1000); //Retard de 1 seconde.
            }
        }
```

L'étape suivante consiste à effectuer la simulation dans ISIS, pour cet effet, les dispositifs suivants sont découverts : 16F84A, RES, et 7SEG-COM-CATHODE. Pour la liaison entre le PIC et l'afficheur, on devrait utiliser les résistances de 330Ω. Par la suite, on réalise le circuit suivant :

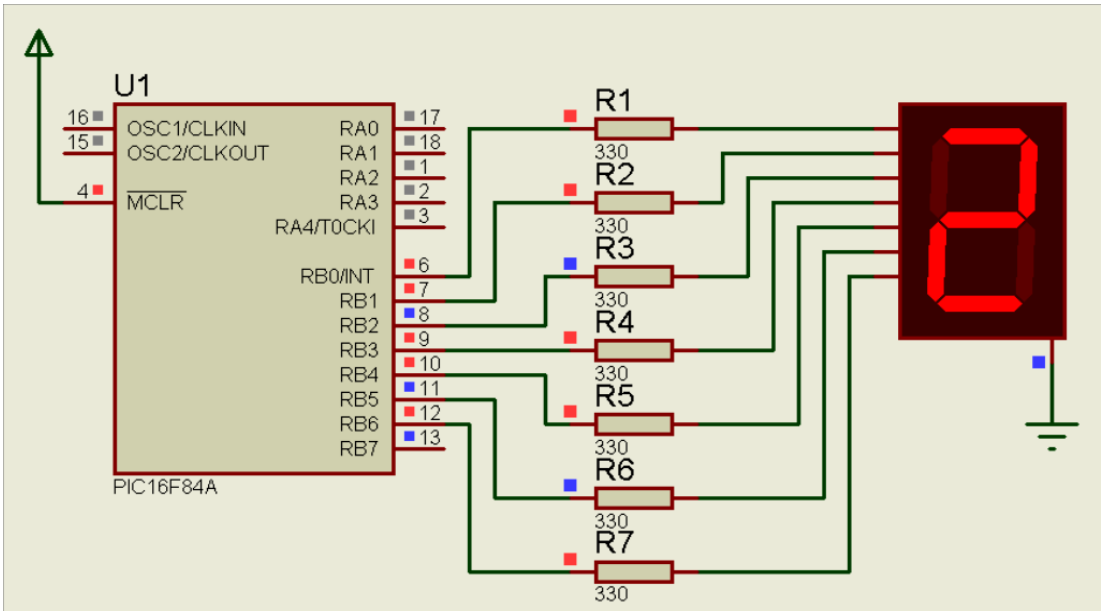

Figure 4

A l'exécution de la simulation, on doit voir sur l'afficheur un comptage des nombres de 0 à 9, avec une cadence d'une seconde entre chiffres et chiffre.

2. Contrôle de l'affichage dynamique 7 segments

La mise en œuvre des afficheurs dynamiques utilise la même théorie d'un seul afficheur. L'affichage dynamique consiste à afficher un seul chiffre à la fois. Par exemple, si on veut afficher quatre chiffres, on active l'afficheur des unités, ensuite on l'éteint et on active le chiffre des dizaines, puis on l'éteint et le chiffre des centaines est activé, enfin on fait la même chose avec l'afficheur des milliers.

Ce processus doit être effectué à une vitesse de manière à tromper l'œil humain, de sorte à voir comme si tous les chiffres ont été actifs. Cette gestion d'affichage minimise les connexions électriques et la consommation d'énergie, car en réalité un seul afficheur est actif à la fois.

Vue la perception de l'œil humain, les modifications doivent être de 25Hz ou plus. Donc, tous les chiffres doivent être montrés pendant une durée égale à l'inverse de 25Hz qui est 40ms.

Pour cet exemple, on utilise 4 afficheurs donc le temps d'affichage de chaque chiffre doit être un quart de la période qui est 10ms. Le moyen le plus efficace pour afficher les chiffres est d'utiliser une fonction.

On réalise la simulation sur ISIS avec les dispositifs suivants 16F84A, RES, 7SEG-MPX4-CC,7404. Le circuit est le suivant :

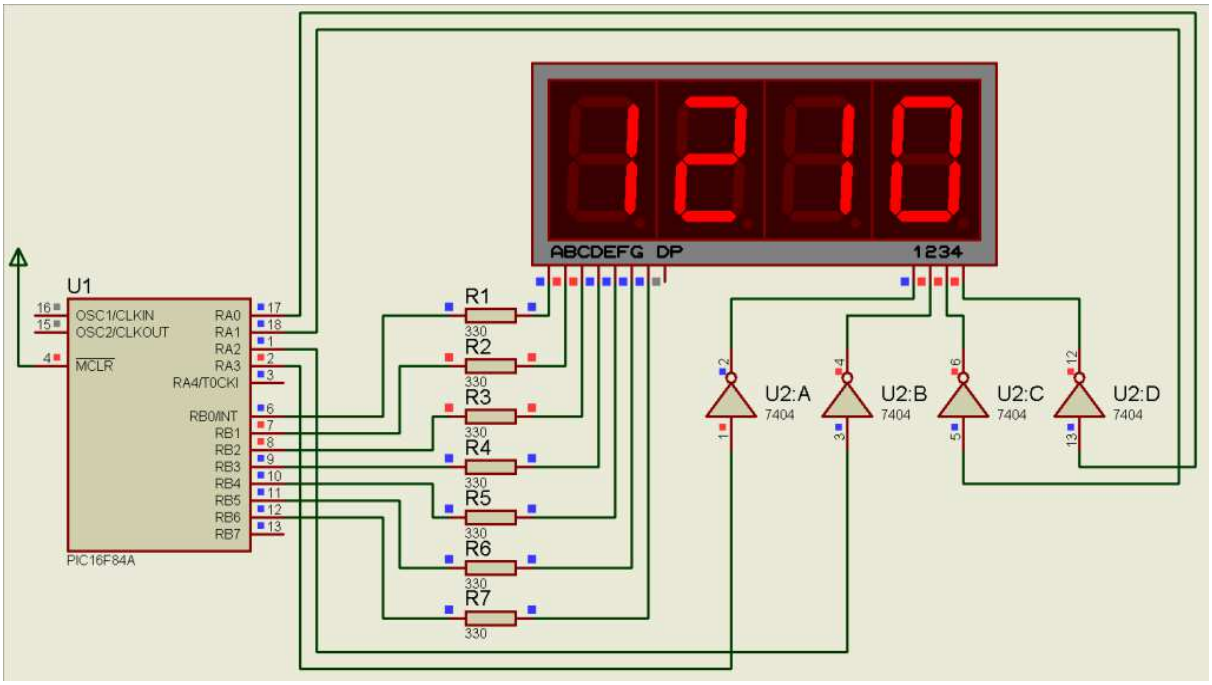

Figure 5

Pour des raisons pratiques, les inverseurs 7404 peuvent être remplacés par un réseau de transistors, comme on le voit dans l'image ci-dessous :

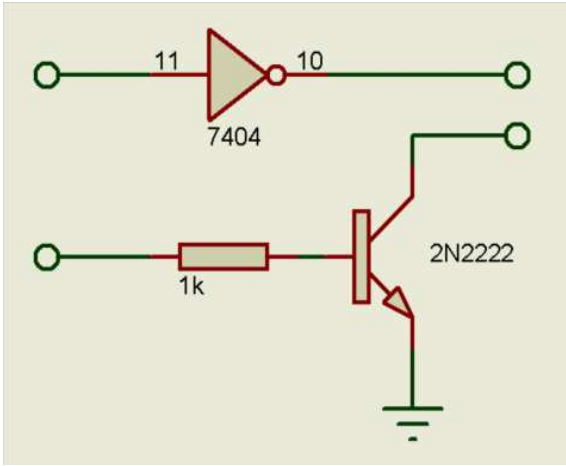

Figure 6

Lorsque la simulation est en marche, ce circuit devrait afficher un nombre de 0 à 9999 avec une cadence de 500ms entre chaque incrément.

Avec 4 digits, il est possible d'afficher un nombre de 0 à 9999, à ce nombre on associe une variable de type entier. Pour voir chaque chiffre séparément, il est nécessaire de calculer chacun des chiffres, par exemple pour déduire le chiffre des milliers on divise le nombre par mille, puis soustraire les

milliers du nombre entier, ce processus est répété jusqu'à atteindre les unités. L'activation des afficheurs doit se faire par un autre port, dans cet exemple elle est faite par le port A.

Pour comprendre ce processus, on observe et on analyse la fonction suivante :

```
// Déclaration des constantes pour l'afficheur.
const unsigned short DIGITS[] =
{
        0x3F, //Code du digit 0
        0x06, //Code du digit 1
        0x5B, //Code du digit 2
        0x4F, //Code du digit 3
        0x66, //Code du digit 4
        0x6D, //Code du digit 5
        0x7D, //Code du digit 6
        0x07, //Code du digit 7
        0x7F, //Code du digit 8
        0x6F, //Code du digit 9
};
//Fonction pour l'un affichage dynamique.
void Affichage( int Nombre )
{
        unsigned short U; //Variable pour les unités.
        unsigned short D; //Variable pour les dizaines.
        unsigned short C; //Variable pour les centaines.
        unsigned short M; //Variable pour les milliers.
        M = Nombre/1000; //Calcul des milliers.
        C = (Nombre-M*1000)/100; //Calcul des centaines.
        D = (Nombre-M*1000-C*100)/10; //Calcul des dizaines.
        U = (Nombre-M*1000-C*100-D*10); //Calcul des unités.
        PORTB = DIGITS[U]; //Visualisation des unités.
        PORTA.F0=1; //Activer le premier afficheur
        delay_ms(10); //Retard de 10ms
        PORTA=0; //Désactiver tous les afficheurs.
```

```
        PORTB = DIGITS[D]; // Visualisation des dizaines.
        PORTA.F1=1; //Activer le second afficheur
        delay_ms(10); // Retard de 10ms
        PORTA=0; // Désactiver tous les afficheurs.
        PORTB = DIGITS[C]; //Visualisation des centaines.
        PORTA.F2=1; //Activer le troisième afficheur
        delay_ms(10); // Retard de 10ms
        PORTA=0; // Désactiver tous les afficheurs.
        PORTB = DIGITS[M]; //Visualisation des centaines.
        PORTA.F3=1; // Activer le troisième afficheur
        delay_ms(10); // Retard de 10ms
        PORTA=0; // Désactiver tous les afficheurs.
}
void main ( void )
{
        unsigned short N=0; //Variable de comptage.
        int Nombre=0;
        TRISB = 0; //Configurer le port B en sortie
        TRISA = 0; // Configurer le port A en sortie
        PORTA = 0; //Désactiver tous les afficheurs
        while( 1 ) //Boucle infinie
        {
                //Visualiser la valeur du Nombre.
                Affichage(Nombre); //cette fonction dure approximativement 40ms.
                //On compte 12 incrémentations de N pour faire une incrémentation
                //du Nombre approximativement chaque 500ms.
                N++;
                if( N==12 )
                {
                        N=0; //Initialiser le compteur N.
                        Nombre++; //Incrémentation de la valeur Nombre.
                        if(Nombre==10000 ) //Test si Nombre vaut 10000
```

```
                                    Nombre=0; //Initialisation à 0 si nombre = 10000.
                        }
                }
        }
```

III. AFFICHEURS LCD

Les afficheurs de caractères à cristaux liquides LCD sont des modules préfabriqués qui contiennent des pilotes inclus. Ces écrans disposent d'un bus de données et un bus de commande, pour la manipulation de ces appareils, le compilateur MikroC PRO dispose d'une bibliothèque prédéfinie pour contrôler les LCD. L'apparence physique des LCD et celle dans ISIS sont présentées dans la figure suivante :

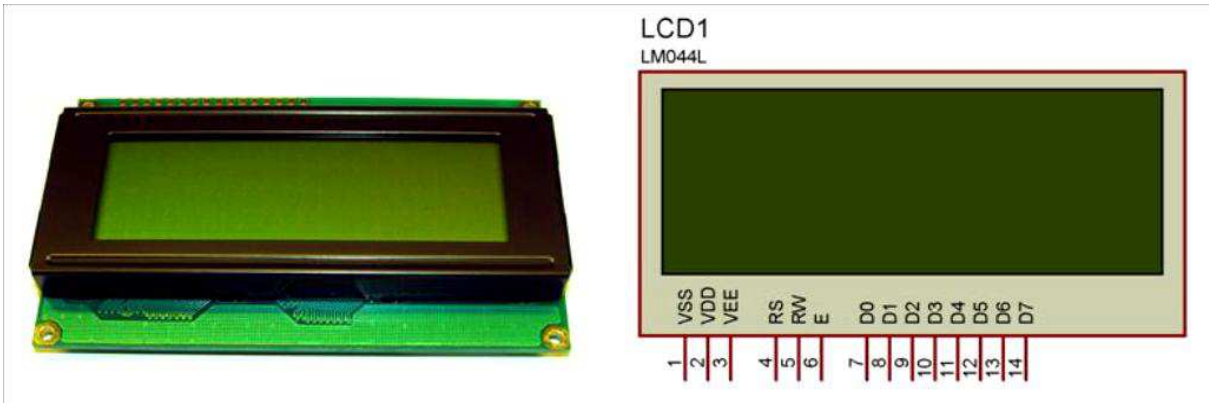

Figure 7

Les afficheurs LCD, permettent de visualiser les caractères figurant dans le code ASCII. En plus du code ASCII, LCD affiche jusqu'à 8 caractères conçus par le développeur. Une autre caractéristique fondamentale des afficheurs LCD, tenant physiquement sur 8 bits, est la possibilité de configurer les connexions avec seulement 4 bits. La connexion 8-bit implique un plus grand nombre de fils à utiliser, pour une meilleure vitesse de travail, par conséquent les connexions de 4 bits réduits le nombre de fils, mais diminue la vitesse. La bibliothèque prédéfinie dans MikroC PRO fonctionne avec une configuration de 4 bits.

Pour voir la bibliothèque prédéfinie pour ce dispositif et d'autres, contenues dans MikroC PRO, on choisit *View* dans la barre de menu, et on appuie sur l'un onglet : *Library Manager*. Lorsqu'on appuie sur cet onglet un menu montrant les différentes bibliothèques qu'on peut utiliser avec les PIC.

Dans ce nouveau menu, on identifie la bibliothèque *Lcd*, par la suite on peut appuyer sur une des fonctions contenues dans la bibliothèque pour voir l'aide. L'aspect visuel de cet outil est illustré dans la figure suivante :

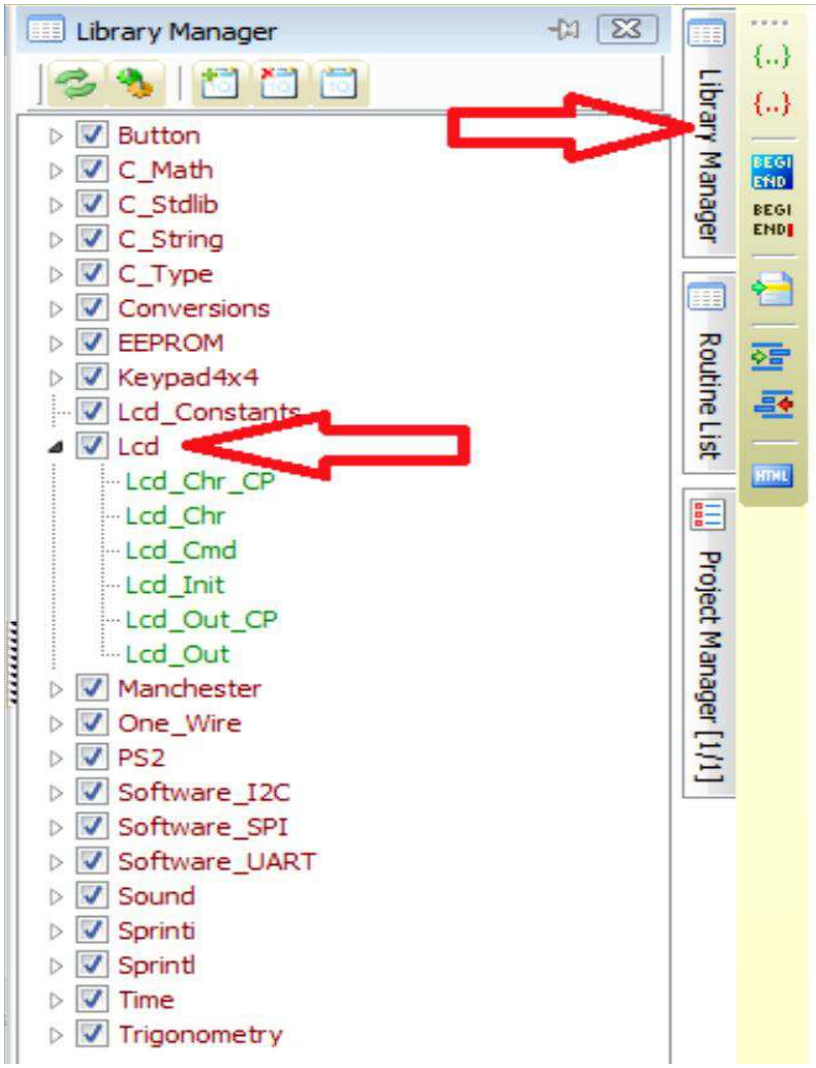

Figure 8

L'implémentation de l'afficheur LCD, requiert l'utilisation des instructions et des commandes séquentielles pour la configuration et l'utilisation de l'afficheur, cependant, la bibliothèque de MikroC PRO minimise ce travail, car elle se charge de faire tous ces réglages, ce qui rend beaucoup plus facile le travail développeur.

Comme première étape pour l'utilisation du LCD, il est nécessaire de définir les broches de connexion, puis l'exécution de la fonction d'initialisation du LCD : *Lcd_Init()*. La définition des broches de connexion est assurée par le développeur d'une façon arbitraire selon son choix. Pour répondre à cet objectif, on utilise la déclaration suivante des constantes :

```
//Broches de sortie du LCD
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D7 at RB3_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D5 at RB1_bit;
```

```
sbit LCD_D4 at RB0_bit;
//Bits de configuration TRIS
sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D7_Direction at TRISB3_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D4_Direction at TRISB0_bit;
```

Pour changer les broches, il suffit de changer les noms des sorties du port utilisé dans l'exemple précédent. Comme on peut le voir dans la présentation ci-dessus seulement 6 broches sont nécessaires pour faire fonctionner le LCD, avec 4 bits de données et 2 bits de contrôle.

A la fin, on doit appeler la fonction d'initialisation dans la fonction *main* après configuration des ports.

La fonction *main* doit être comme suit :

```
void main( void )
{
        Lcd_Init(); //Initialisation du LCD.
        while(1) //Boucle infinie.
        {
        }
}
```

Après l'édition du code ci-dessus, l'afficheur LCD est initialisé et devrait être prêt à commencer l'affichage des caractères, se positionnant dans la première ligne et la première colonne, montrant le curseur clignotant.

Les LCD sont fabriqués dans différentes formes et couleurs, ils sont disponibles avec des écrans verts, bleu et jaune, avec des distributions de caractères sous forme de matrice comme les LCD 2 lignes, 16 colonnes. Ceux-ci sont connus comme 2x16, de la même manière on peut trouver des 1x16, 2x16, 2x8, 2x20, 4x20, entre autres. Pour les exemples de ce chapitre, on utilisera l'afficheur 4x20.

Pour démarrer la simulation de l'afficheur LCD, on cherche le dispositif de LM044L, et PIC 16F84A dans le simulateur ISIS. La référence LM044L dans ISIS correspond à un LCD de 4x20. Enfin, on effectue les connexions comme indiqué dans le circuit suivant :

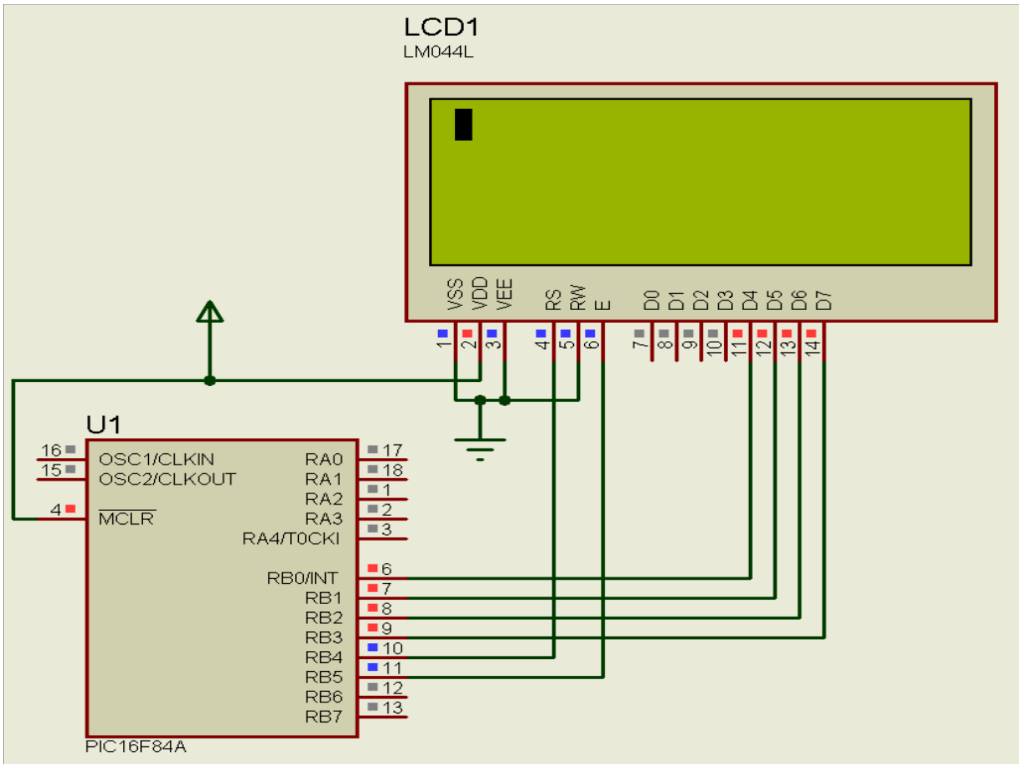

Figure 9

L'afficheur LCD a une broche nommée VEE, cette broche fonctionne comme contrôleur du contraste de l'écran, mais dans la simulation, elle n'a pas d'effet. Cette broche peut être reliée à la masse pour générer le contraste le plus élevé, certains grands écrans nécessitent une tension négative externe pour contrôler le contraste. Pour des raisons pratiques, le contraste peut être ajusté par l'intermédiaire d'un potentiomètre, comme indiqué dans le circuit suivant :

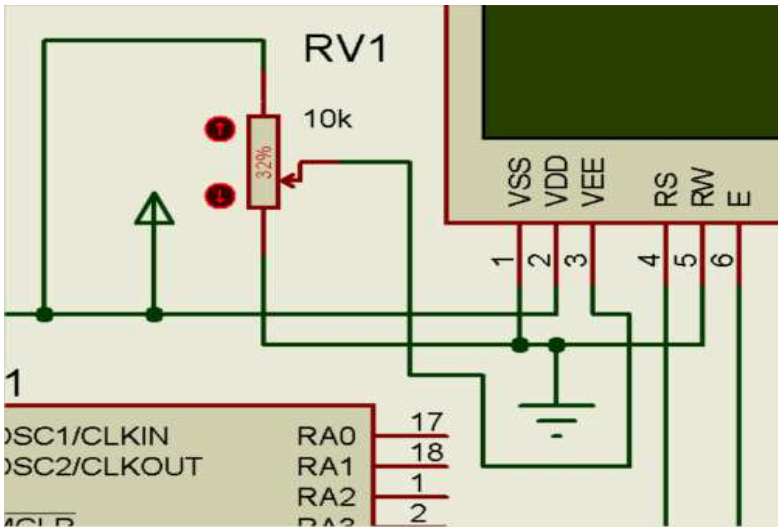

Figure 10

La prochaine étape consiste à afficher à l'écran des informations. A cet effet, on peut utiliser quatre fonctions. Deux de ces fonctions permettent d'afficher des caractères, et les deux autres des chaînes de caractères.

Pour afficher les caractères, on peut procéder de deux façons, la première affiche simplement les caractères dans l'ordre consécutif pris par l'afficheur et la seconde fonction imprime les caractères dans la ligne et la colonne désignées par le développeur.

1. Fonctions d'affichage des caractères

La première fonction d'affichage des caractères est *Lcd_Chr_Cp (char out_char)* ; lorsque cette fonction est invoquée affiche sur l'écran le caractère qui correspond au code ASCII, qui est le paramètre d'entrée *out_char*. Avec l'impression d'un nouvel caractère sur l'afficheur LCD, le curseur se déplace automatiquement d'une position. Pour voir le fonctionnement de cette fonction, on observe l'exemple suivant :

```
void main( void )
{
        Lcd_Init(); //Initialisation du LCD.
        Lcd_Chr_Cp('S'); //cette fonction affiche lettre à lettre le mot "Salut".
        Lcd_Chr_Cp('a');
        Lcd_Chr_Cp('l');
        Lcd_Chr_Cp('u');
        Lcd_Chr_Cp('t');
        while(1) //Boucle infinie.
        {
        }
}
```

Après l'exécution de la simulation, on devrait observer ce qui suit sur l'afficheur LCD :

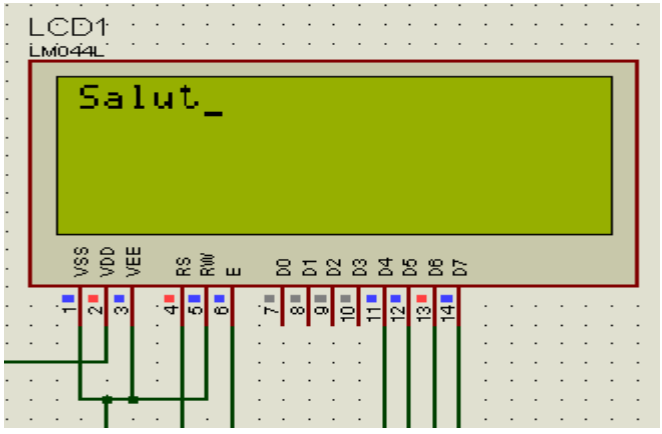

Figure 11

Pour l'affichage des caractères au moyen des cordonnées ligne, colonne, la fonction mise en œuvre : *Lcd_Chr(**char** row, **char** column, **char** out_char);* cette fonction imprime le caractère *out_char*, dans la colonne *column* et la ligne *row*. Dans l'exemple ci-dessous, on peut voir comment utiliser cette fonction :

```
void main( void )
{
    Lcd_Init(); //Initialisation du LCD
    Lcd_Chr_Cp('S'); //Ces fonctions impriment lettre à lettre le mot "Salut".
    Lcd_Chr_Cp('a');
    Lcd_Chr_Cp('l');
    Lcd_Chr_Cp('u');
    Lcd_Chr_Cp('t');
    Lcd_Chr( 1, 6, '1'); //Imprime le caractère 1, dans la ligne 1, colonne 6
    Lcd_Chr( 2, 7, '2'); // Imprime le caractère 2, dans la ligne 2, colonne 7
    Lcd_Chr( 3, 8, '3'); // Imprime le caractère 3, dans la ligne 3, colonne 8
    Lcd_Chr( 4, 9, '4'); // Imprime le caractère 4, dans la ligne 4, colonne 9
    while(1) //Boucle infinie.
    {
    }
}
```

Après l'exécution de la simulation, on devrait observer ce qui suit sur l'afficheur LCD :

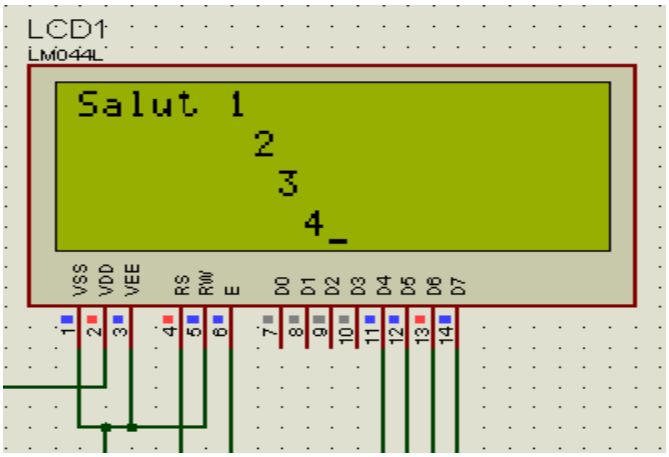

Figure 12

2. Fonctions d'affichage des chaînes de caractères

L'utilisation de chaînes est similaire aux deux fonctions précédentes, pour cela on peut afficher des chaines de caractères à un point arbitraire à l'aide des coordonnées ligne et colonne. Pour imprimer une chaîne de caractères à la position courante du curseur, on utilise la fonction suivante : *Lcd_Out_Cp (char * text)* ; elle a un paramètre d'entrée unique qui est un pointeur vers la chaîne. Pour voir comment utiliser cette fonction, on observe l'exemple suivant :

```
void main( void )
```

```
{
        Lcd_Init(); //Initialisation du LCD.
        Lcd_Out_Cp("salut tout le Monde");
        while(1) //Boucle infinie.
        {
        }
}
```

La mise en œuvre de cette fonction peut être faite avec des chaînes de caractères constantes ou variables, des chaines constantes sont indiquées par des guillemets au début et à la fin du texte, par exemple "Salut tout le Monde ", la forme variable est déclarée : char texte [20] = "Salut tout le Monde ". Après avoir exécuté la simulation, on a comme montré dans la figure suivante :

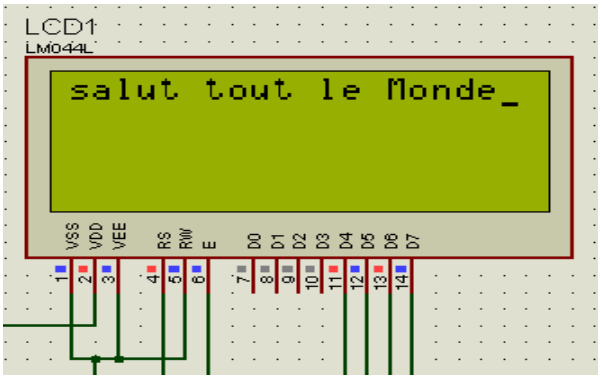

Figure 13

Pour imprimer une chaîne de caractères avec une coordonnée comme point de départ, on implémente la fonction : *Lcd_Out(**char** row, **char** column, **char** *text);*. Cette fonction est similaire à la précédente, à la différence qu'on inclut les données *row* et *column*, qui font référence respectivement à la ligne et la colonne. Pour bien comprendre le fonctionnement de cette fonction d'observe et on analyse l'exemple suivant :

```
void main( void )
{
        Lcd_Init(); //Initialisation du LCD.
        Lcd_Out( 1, 1, "Ligne 1, Colonne 1" );
        Lcd_Out( 2, 2, "Ligne 2, Colonne 2" );
        while(1) //Boucle infinie.
        {
        }
}
```

Après la compilation et la simulation de cet exemple, on devrait observer sur l'écran LCD comme suit:

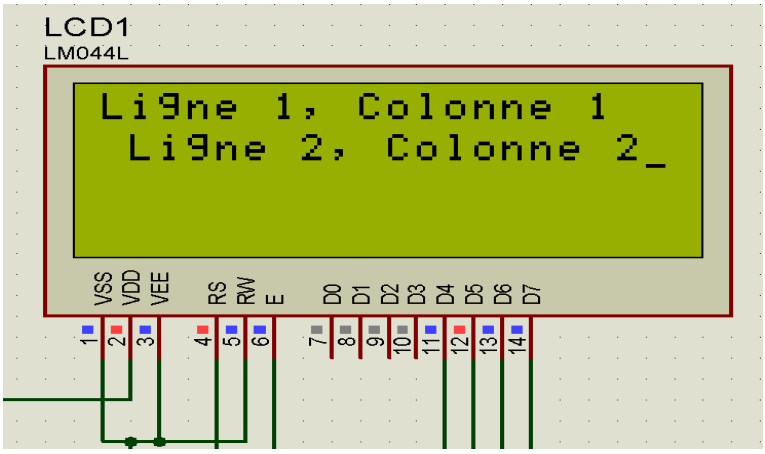

Figure 14

3. Affichage des valeurs numériques

La visualisation des valeurs numériques est indispensable dans de nombreux développements. Par exemple, lorsqu'on souhaite afficher l'état d'une variable, ou un capteur tel que : température, humidité, la pression, etc.

Pour atteindre cet objectif, on peut recourir aux fonctions de conversion prédéfinies par le compilateur. Pour cela, on peut utiliser la bibliothèque : *Conversions,* qui contient des fonctions qui convertissent une valeur numérique en une chaîne de caractères.

Si on veut afficher une valeur de type entier, on utilise la fonction *: IntToStr(**int** input, **char** *output)*; cette fonction a deux paramètres d'entrée sont: *input*, qui est une valeur entière à afficher, et *output*, qui est un pointeur sur une chaîne où on veut écrire la forme texte de la valeur *intput.* Pour comprendre ce type de conversion, on observe et on analyse l'exemple suivant :

```
void main( void )
{
    int ENTIER=123; //Déclaration d'une variable entière avec valeur initiale 123.
    char Text[20]; //Chaine de caractères pour l'affichage des données.
    Lcd_Init(); //Initialisation du LCD.
    IntToStr( ENTIER,Text ); //Fonction de conversion.
    Lcd_Out_Cp(Text); //Impression du texte dans l'écran LCD.
    while(1) //Boucle infinie.
    {
    }
}
```

L'impression des nombres entiers en chaîne de caractères avec cette fonction, réserve toujours un champ fixe de 7 caractères, c'est à dire si le nombre est inférieur à 7 chiffres le reste du texte est complété par des espaces vides. Après compilation du programme et simulation sur ISIS, on a un résultat comme dans la figure suivante :

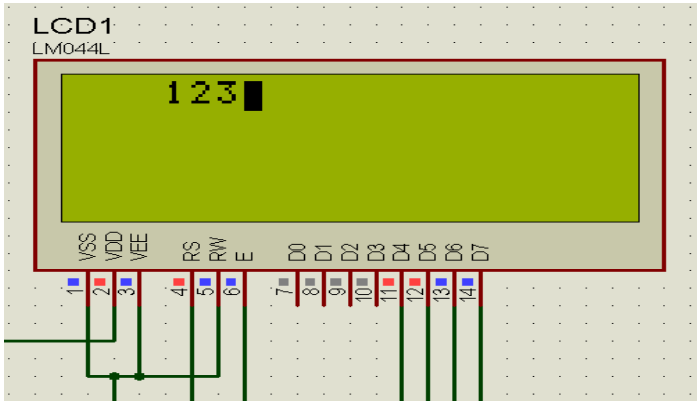

Figure 15

L'impression des nombres décimaux (avec virgule) peut être fait avec la fonction : *FloatToStr (**float** fnum, **char** *str);*. La philosophie de fonctionnement de cette fonction est identique à la précédente, qui fait des conversions d'entiers. Pour réaliser l'exemple suivant, on doit modifier la référence du microcontrôleur, cela est dû à la capacité de mémoire du PIC 16F84A, qui n'est pas suffisante pour les exercices suivants.

```
void main( void )
{
    int ENTIER=123; //Déclaration d'une variable entière avec la valeur initiale 123.
    float DECIMAL=12.76543; //Déclaration d'une variable avec point décimal
                            //initialisation à 12,76543.
    char Text[20]; // Chaine de caractères pour l'affichage des données.
    Lcd_Init(); //Initialisation du LCD.
    IntToStr( ENTIER,Text ); //Fonction de conversion entière .
    Lcd_Out(1,1,Text); //Impression du texte en sur l'écran LCD.
    FloatToStr( DECIMAL,Text ); //Fonction de conversion décimale.
    Lcd_Out(2,1,Text); // Impression du texte en sur l'écran LCD.
    while(1) //Boucle infinie.
    {
    }
}
```

Après la simulation, on a le résultat sur la figure suivante :

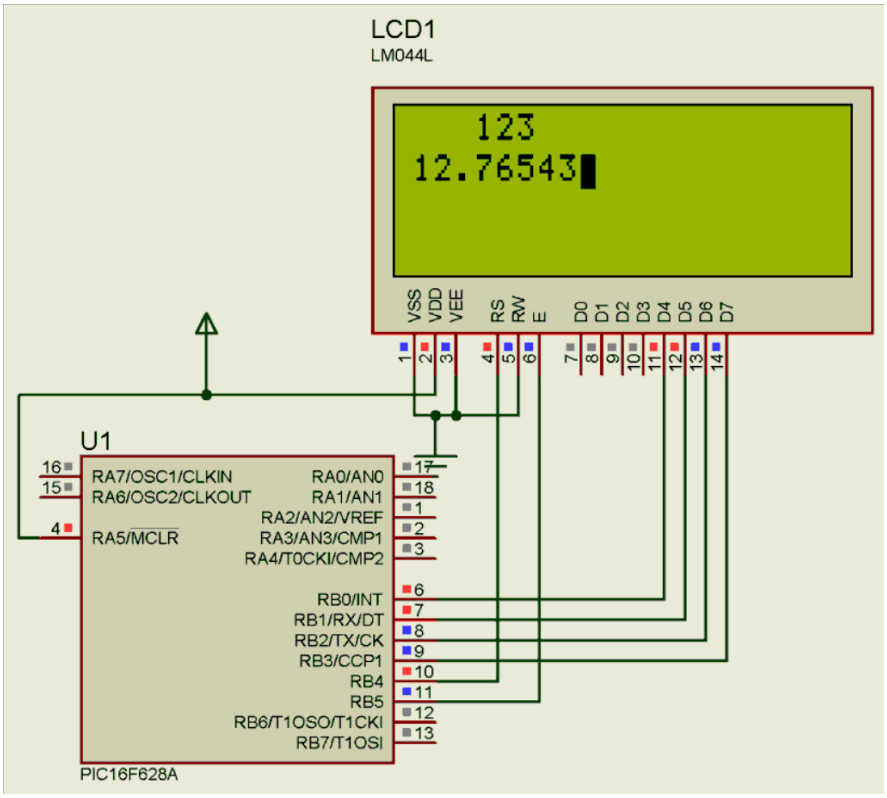

Figure 16

Le même processus peut être suivi pour autres types de variables telles que : *short* avec la fonction : *ShortToStr(**short** input, **char** *output);*

Les variables *long* avec la fonction : *LongToStr(**long** input, **char** *output);*

Des variables *unsigned short* avec la fonction : *ByteToStr(**unsigned short** input, **char** *output);*

Variables *unsigned long* avec la fonction : *LongWordToStr(**unsigned long** input, **char** *output);*

Les variables *unsigned int* avec la fonction : *WordToStr(**unsigned int** input, **char** *output);*