Aprendiendo C18

Introducción a la Programación y uso

V3.2 August 10, 2007

Agenda

- Instalacion de C18
- Construyendo nuestro Primer Proyecto
- Arquitectura de un Programa en C18
- Tipos de datos y variables
- Almacenamiento en Memoria de Datos y Memoria de Programa
- Introducción a las Funciones
- Visibilidad de las variables
- Declaración de variables
- Como escribir y leer un PORT

Agenda cont.

- Estructuras condicionales IF
- Bucles:
  - Bucle FOR
  - Bucle WHILE
  - Bucle DO-WHILE
  - Break , Contine y Goto
- Operaciones logicas
  - Operadores logicos en condicionales
  - Operadores logicos a nivel BIT

Agenda cont.

- Arrays
- Funciones
  - Modularizacion de un programa
  - Creacion y declaracion de funciones
  - Llamada a funciones
  - Entrega de parametros
- Directivas
  - #include
  - #define

Agenda cont.

- Las Librerias de C18
  - Librerias para el manejo de Perifericos
    - Manejo de la USART
    - Manejo del Timers
    - Manejo del ADC
  - Librerias para control por software
    - Manejo de displays LCDs
  - Librerias para tratamiento de Strings

Instalacion de C18

V3.2 August 10, 2007

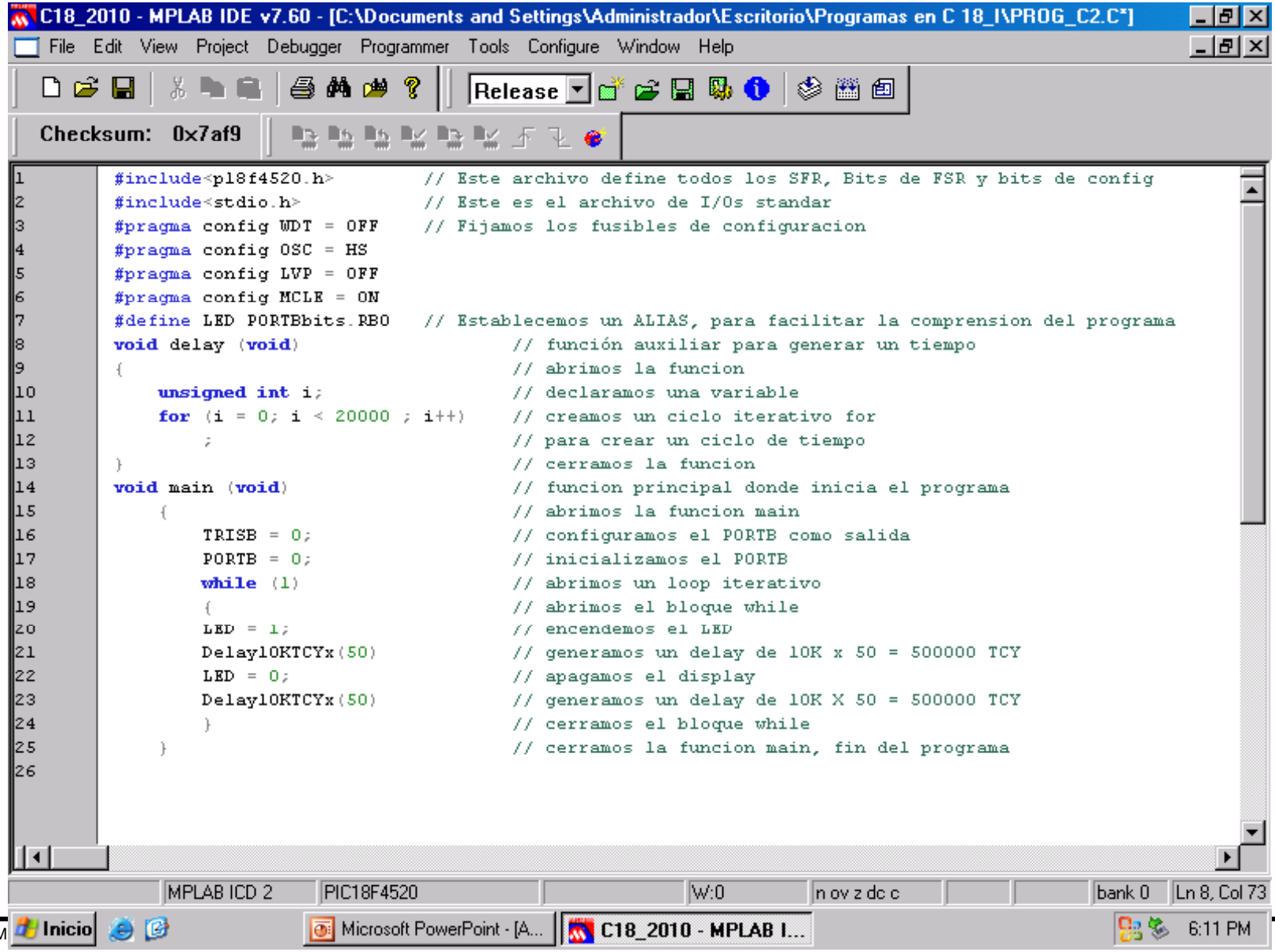
Como hacer nuestro Pimer Proyecto

V3.2 August 10, 2007

Arquitectura de un Programa en C18

V3.2 August 10, 2007

Arquitectura de un programa en C18



Tipos de Datos en MPLAB C18

- **Los datos que puede manejar C pueden ser del tipo enteros, enteros positivos y negativos, decimales con presición simple o mayor, llamados “flotantes de simple y doble presición”**

Type	Size	Minimum	Maximum
`char`[1,2]	8 bits	-128	127
`signed char`	8 bits	-128	127
`unsigned char`	8 bits	0	255
`int`	16 bits	-32,768	32,767
`unsigned int`	16 bits	0	65,535
`short`	16 bits	-32,768	32,767
`unsigned short`	16 bits	0	65,535
`short long`	24 bits	-8,388,608	8,388,607
`unsigned short long`	24 bits	0	16,777,215
`long`	32 bits	-2,147,483,648	2,147,483,647
`unsigned long`	32 bits	0	4,294,967,295

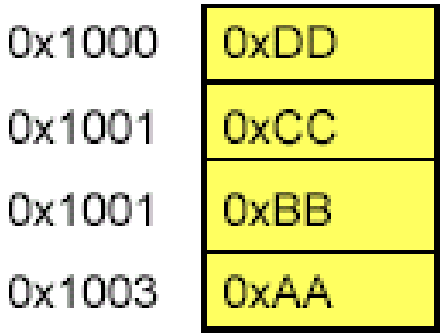
Datos y Variables

V3.2 August 10, 2007

Formato de datos Little Endian

Los datos de más de un byte de longitud, se almacenan en memoria siguiendo el criterio LITTLE ENDIAN, es decir los bytes menos significativos ocupan las posiciones de memoria más bajas.
Ejemplo:

#pragma idata mi_dato=0x1000
Long valor = 0xAABBCCDD;



Las variables

- Para almacenar los datos dentro de un microcontrolador se usa la memoria, el lugar donde se pueden almacenar temporalmente los mismos, los denominamos VARIABLES
- Desde el punto de vista físico la variable se aloja en un registro, el cual puede almacenar 8 bits. Sin embargo una variable puede ocupar mas de un registro, todo depende del largo que tenga el dato que quiera almacenarse en la variable
- Las variables por tanto se identifican por un nombre o “identificador” y por tener una dimensión lo cual se asigna mediante un “tipo” que puede ser modificado por un “calificador”

Tipos de Variables: Char

- Para almacenar datos las variables podrán ser:
  - char (carácter): puede contener un carácter o un dato numérico de -128 a +127
  - unsigned char (carácter sin signo): puede contener un dato de 0 a 255
  - signed char (carácter con signo): es igual que la tipo char

Registro

Variables tipo int

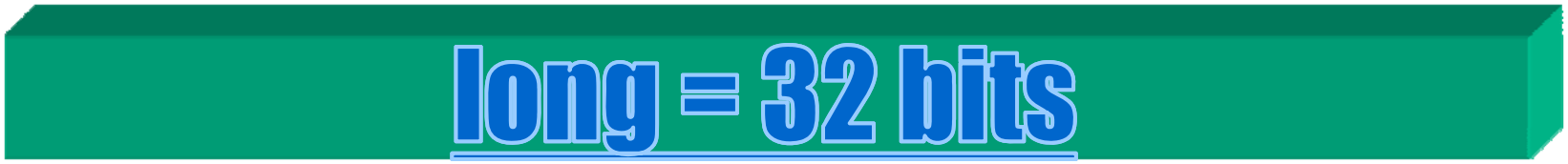
- La variable tipo int ocupa 2 bytes (16 bits), y la misma se usa típicamente para almacenar datos numéricos.
  - Int se usa cuando queremos almacenar cualquier número que vaya desde -32768 a +32767
  - Unsigned int permite el rango máximo de almacenamiento pero solo para números positivos desde 0 a 65535

Variables tipo short

- La variable tipo Short ocupa 2 bytes como lo hace int (16 bits), pero si le sigue el modificador long ampliamos su rango a 3 bytes (24bits)
  - short se usa cuando queremos almacenar cualquier número que vaya desde -32768 a +32767
  - unsigned short permite el rango máximo de almacenamiento pero solo para números positivos desde 0 a 65535
  - short long permite un rango mayor de almacenamiento con un rango negativo y positivo de números que va desde -8.368.608 a +8.368.607 (24 bits)
  - unsigned short long permite todo el rango pero solo admite numeros positivos desde 0 a 16.777.215 (24 bits)

Variables tipo long

- La variable tipo long ocupa 4 bytes, se usa cuando queremos almacenar cualquier número que vaya desde -2.147.483.648 a +2.147.483.647
- unsigned long permite el rango máximo de almacenamiento pero solo para números positivos desde 0 a 4.294.967.295



b31

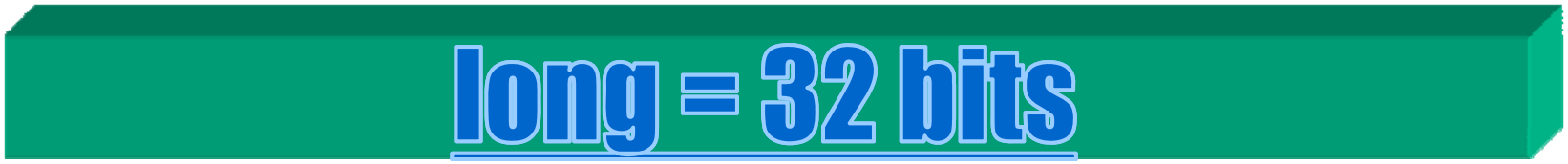
4 Registros de 8bits

b0

Variables tipo float y double

- Las variables tipo **float** y **double** ocupan 4 bytes, se usan cuando queremos almacenar cualquier número decimal.

Type	Size	Minimum Exponent	Maximum Exponent	Minimum Normalized	Maximum Normalized
`float`	32 bits	-126	128	$2^{-126} \approx 1.17549435e-38$	$2^{128} * (2-2^{-15}) \approx 6.80564693e+38$
`double`	32 bits	-126	128	$2^{-126} \approx 1.17549435e-38$	$2^{128} * (2-2^{-15}) \approx 6.80564693e+38$



b31

4 Registros de 8bits

b0

Almacenamiento de variables

- Las variables pueden crearse en la memoria de datos (ram) o en la memoria de programa del microcontrolador (rom).
- Para indicarle al compilador donde deberá crear la variable se antepone a la definición de la variable las siglas rom o ram.
- Sino indicamos cual es el lugar donde se almacenaran las variables, estas por default se almacenaran en memoria de datos

Almacenamiento en memoria de Programa

- Para almacenar datos en memoria de programa eixsten 2 modificadores, denominados near y far (cercana y lejana:
  - far: cuando se usa este modificador la variable se crea en la memoria de programa por encima de los 64K
  - near: cuando se usa este modificador la variable se crea en memoria de programa por debajo de los 64K
  - Ejemplo:

```
#define MEM_MODEL near
```

Las funciones

- Un programa de C esta constituido por rutinas, o segmentos de código que realizan una función determinada.
- Las funciones tienen un nombre, mediante el cual se identifican.
- La función mas importante, y donde comienza todo programa se denomina main (principal)
- Todo programa debe tener una función main donde se encuentra la parte principal del mismo.
- Las funciones son bloques de sentencias de código, y están limitadas por llaves {}. Toda función comienza con el "{" y termina con " } "

Visibilidad de las variables

- Las funciones usan las variables, las cuales si son creadas dentro de las mismas se denominan **locales**, mientras que si son creadas fuera de las funciones, se denominan **globales**.
- Las variables **locales** solo existen dentro de las funciones donde fueron creadas, y son destruidas, al terminar la función
- Las variables **locales** solo pueden ser vistas y por tanto usadas, dentro de la función donde se crearon.
- Las variables **globales** son vistas por todas las funciones, y por tanto pueden ser usadas por cualquier función.

Ejemplos de definiciones de variables

- Definiendo una variable:

```
int contador;
char buffer;
```

- Definiendo variables múltiples:

```
char buffer1, datos, contador;
```

- Definiendo e inicializando las variables:

```
unsigned long contador = 0;
```

Nota: todas las sentencias en los programas en C terminan siempre con un ; el programador debe tener cuidado en no olvidar esta regla pues generará un error en el compilador

Sistemas de numeración

- Los datos en los programas pueden escribirse en distintos formatos:
  - Decimal: es el formato normal por default
  - Binario: se debe anteponer 0b : 0b1011001
  - Hexadecimal: se antepone 0x:

    0x0F1A

Ejemplo de un Programa que controla un LED

```
#include <p18f4520.h>                          //incluimos el archivo de etiquetas de pines
#pragma config OSC = XT                        //configuramos los fusibles de configuración
#pragma config PWRT = ON
#pragma config WDT = OFF
#pragma config MCLRE = ON
#pragma config STVREN = OFF
#pragma config LVP = OFF
//*****************************************************************************
void main(void)                                //declaramos la función main
{                                              //Abrimos la función main
//*****************************************************************************
    ADCON1=0x0F;                               //Todos entrada/salida digitales.-
    TRISA=0xFF;                                //Todos como entrada.-
    TRISB=0X00;                                //Todos como salida.-
//*****************************************************************************
// Loop principal
//*****************************************************************************
    while (1)                                  //Creamos un loop iterativo infinito
        {                                      //abrimos el loop
            PORTBbits.RB0 = PORTAbits.RA0;     //escribimos en el PORTB, pin RB0
                                               //lo que hay en el PORTA, pin RA0
        }                                      //cerramos el loop
}                                              //cerramos la función main
```

Estructuras dentro de un programa

V3.2 August 10, 2007

Estructuras condicionales

- Las estructuras condicionales evalúan una condición determinada por el programador. La condición a evaluar se coloca entre paréntesis.
- La sentencia para el condicional comienza con la palabra **if**
- Si la condición se cumple, se ejecuta la sentencia siguiente al condicional; caso contrario, se ejecuta la que se encuentra luego de la palabra **else**

Ejemplo:

```
            if(PORTAbits.RA4==1)    // testeamos RA4.
                    PORTBbits.RB0=1;        // Si esta en 1
Encendemos el PORT RB0
                    else
                            PORTBbits.RB1=0;        // Si esta en 0
Apagamos el PORT RB0
```

Sentencias Condicionales cont.

- **Si la condición debe ejecutar mas de una sentencia, estas deben ir entre llaves:**

```
if (PORTAbits.RA4==1)                          // testeamos si RA4 = 1.
                {
                        PORTBbits.RB0=1;       // Si esta en 1
  Encendemos RB0
                                PORTCbits.RC0=0;       // Apagamos RC0
                        contador++;    // incrementamos el contador
                }
        else
                {
                                PORTBbits.RB0=0;       // Sino Apagamos
  RB0.
                        PORTCbits.RC0 = 1; // Encendemos RC1
                        contador--;     // decrementamos el contador
                }
```

Ciclos Iterativos

- Un ciclo iterativo es una secuencia de instrucciones que se repiten sucesivamente hasta que se cumpla una condición.
- Existen varias formas de crear un ciclo iterativo:
  - Ciclo FOR
  - Ciclo While
  - Ciclo Do-While

Ciclos FOR

- Para crear este ciclo se usa la palabra clave FOR
- La condición a evaluar tiene 3 parámetros, los cuales se colocan entre paréntesis y separados por “;”

  (i=0;i<10;i++): en este caso i es la variables que se testea, se inicia en 0, y se evalúa que la misma tenga un valor <10, finalmente indicamos que i se incremente cada vez

Ejemplo:

```
for (i=0 ; i<10 ; i++)
        {
                PORTB = i; // enviamos al PORTB el valor de i
        }
```

Ciclos while

- **Otra forma de un ciclo iterativo es la del while (mientras). En este caso mientras la condición de evaluación se cumpla, se ejecutaran las sentencias encerradas entre llaves. La condición de evaluación debe ir entre paréntesis.**

```
while (PORTAbits.RA0 == 1) // evaluamos que se cumpla RA0 = 1
        {                          // abrimos bloque que se ejecutará
                contador++;        // incrementamos el contador
                PORTB = contador; // mostramos el contador
        }                          // cerramos bloque
```

Ciclo do-while

- **El ciclo do-while es similar al while, con la diferencia que mientras que en el while, si la condición no se cumple, el bloque de sentencias no se ejecuta; en el do while por lo menos se ejecutan una vez.**
- **La condición es checkeada al final del bloque**

```
do {
        contador ++;
        PORTB = contador;
} while (PORTAbits.RA1 = 1);
```

Sentencias de control BREAK

Sentencia *BREAK*

• interrumpe la ejecución de un bucle **while**, **do-while** o **for**.

Ejemplo: ¿ cómo salir de un bucle infinito **for** ?

```
For (;;) {
        if ( a == 0) {
                break;
                  }
        a=PORTA;
        }
```

Sentencia de Control CONTINUE

Sentencia *CONTINUE*

- Se utiliza en los bucles para pasar a la siguiente repetición.

```
For (;;) {
        if ( PORTAbits.RA0 = 1) {
                continue;
                }
        contador++;
        }
```

Sentencias de Control
GOTO

Sentencia goto

Transfiere incondicionalmente el control a la sentencia etiquetada por el identificador.

goto *identificador;*

identificador:
instrucciones;

¡¡¡ No es una buena práctica utilizarla !!!

Operadores

V3.2 August 10, 2007

Operadores de Relación

Operador	Operación	Ejemplo	Resultado (FALSE= 0, TRUE ≠ 0)
<	Menor que	`x < y`	1 si `x` menor que `y`, sino 0
<=	Menor que o igual a	`x <= y`	1 si `x` menor o igual a `y`, sino 0
>	Mayor que	`x > y`	1 si `x` mayor que `y`, sino 0
>=	Mayor que o igual a	`x >= y`	1 si `x` mayor o igual a `y`, sino 0
==	Igual a	`x == y`	1 si `x` igual a `y`, sino 0
!=	No igual a	`x != y`	1 si `x` no igual a `y`, sino 0

En expresiones condicionales, cualquier valor no 0 es es interpretado TRUE. Un valor de cero 0 es siempre FALSE.

Operadores de relación

Diferencia entre = y ==

Tenga cuidado de no confundir = y ==.
¡No son intercambiables!

- **= es el operador asignado**

  `x = 5` asigna el valor 5 a la variable `x`

- **== es el operador relacional 'igual a'**

  `x == 5` prueba si es el valor de x es **5**

```
if (x == 5)
{
    haga si el valor de x es 5
}
```

Operadores Lógicos

Operador	Operacion	Ejemplo	Resultado (FALSE = 0, TRUE ≠ 0)
`&&`	Logica AND	`x && y`	1 Si ambos `x` ≠ 0 y `y` ≠ 0, sino 0
`||`	Logica OR	`x || y`	0 Si ambos `x` = 0 e `y` = 0, sino 1
`!`	Logica NOT	`!x`	1 if `x` = 0, sino 0

En expresiones condicionales, cualquier valor no zero es interpretado como TRUE. Un valor 0 es siempre FALSE.

Operadores Lógicos a nivel BIT (Bitwise)

Operador	Operación	Ejemplo	Resultado (para cada bit)
`&`	Bitwise AND	`x & y`	1, si x e y son 1 0, si x o y son 0
`|`	Bitwise OR	`x | y`	1, si x o y son 1 0, si x e y 0 son 0
`^`	Bitwise XOR	`x ^ y`	1, si x es distínto de y 0, si x es igual a y
`~`	Bitwise NOT (Complemento a uno)	`~x`	1, si x es 0 0, si y es 1

- **La operación se realiza Bit a Bit entre los 2 operandos que estan afectados por la sentencia**

Operadores

Desplazamiento

Operador	Operación	Ejemplo	Resultado
<<	Desplaza a la Izq	`x << y`	Shift `x` por `y` bits a la Izq
>>	Desplaza a la Derecha	`x >> y`	Shift `x` por `y` bits a la Derecha

Ejemplo de desplazamiento a la Izquierda:

```
x = 5;          // x = 0b00000101 = 5
y = x << 2;     // y = 0b00010100 = 20
```

- Cada vez que se ejecuta la sentencia de desplazamiento, el contenido de la variable se desplaza hacia la derecha o la izquierda N cantidad de bits, segun se exprese
- Para desplazar, la variable es rellenada con ceros (no es rotación, es desplazamiento)

Operadores

El operador condicional

- El operador condicional puede ser usado para condicionar el asignarle un valor a una variable

Ejemplo 1 (el mas usado comunmente)

```
x = (condicion) ? a : b;
```

Ejemplo 2 (el menos usado)

```
(condicion) ? (x = a):(x = b);
```

En ambos casos:

x = a si la condicion es true
x = b si la condición es false

Arrays

V3.2 August 10, 2007

Arrays

Definición

Arrays son las variables que pueden almacenar muchos artículos del mismo tipo. Los items individuales son conocidos como **elementos**, son almacenados secuencialmente y son identificados unicamente por el **indice** del ARRAY(llamado aveces un **subscript**).

- **Arrays:**
  - **Pueden contener cualquier número de elementos**
  - **Los elementos deben ser del mismo tipo**
  - **El índice base es cero**
  - **El tamaño del Array (numero de elementos) debe estar en la declaración**

Arrays

Como crear un Array

Los arrays se declaran como las variables:

Sintaxis

```
tipo nombrearray[tamaño];
```

- *tamaño* refierre al número de elementos
- *tamaño* debe ser una constante entera

Ejemplo

```
int a[10];     // Un array que contiene 10 elementos

char s[25];    // Un array que contiene 25 caracteres
```

Arrays

Como inicializar un array en la declaración

Los Arrays pueden ser inicializados con una lista en la declaración :

Sintaxis

```
tipo nombrearray[tamaño] = {item1,…,itemn};
```

- **Los items deben ser todos iguales en el *tipo* de array**

Ejemplo

```
int a[5] = {10, 20, 30, 40, 50};

char b[5] = {'a', 'b', 'c', 'd', 'e'};
```

Arrays

Como usar un array

Arrays son accesibles como las variables, pero con un index:

Sintaxis

```
nombrearray[indice]
```

- *Indice* puede ser una variable o una constante
- El primer elemento en un array tiene un indice 0
- C no proporciona ninguna comprobación de los límites

Ejemplo

```
int i, a[10];   //un array que contiene 10 elementos

for(i = 0; i < 10; i++) {
  a[i] = 0;   //Inicializa los elementos del array a 0
}
a[4] = 42;   //cargamos el elemento 4
```

Arrays

Creando Arrays multidimensionales

Agregue las dimensiones adicionalesen la declaración delarray :

Sintaxis

tipo nombrearray[*tamaño*$_1$]...[*tamaño*$_n$];

- Los Arrays pueden tener cualquier número de dimensiones
- Tres dimensiones tienden a ser las más usadas en práctica común

Ejemplo

```
int a[10][10];          //10x10 array para 100 enteros

float b[10][10][10];//10x10x10 array para 100 decimales
```

Arrays

Creando Arrays multidimensionales

Arrays pueden ser inicializados con un listado dentro de un a listado:

Sintaxis

```
tipo nombrearray[tamaño0]…[tamañon] =
                        {{item,…,item},
                                ⋮
                         {item,…,item}};
```

Ejemplo

```
char a[3][3] = {{'X', 'O', 'X'},
                {'O', 'O', 'X'},
                {'X', 'X', 'O'}};

int b[2][2][2] = {{{0, 1},{2, 3}},{{4, 5},{6, 7}}};
```

Arrays

Visualizando Arrays de 2 dimensiones

```
int a[3][3] = {{0, 1, 2},
               {3, 4, 5},
               {6, 7, 8}};
```

fila, Columna

a[y][x]

fila 0

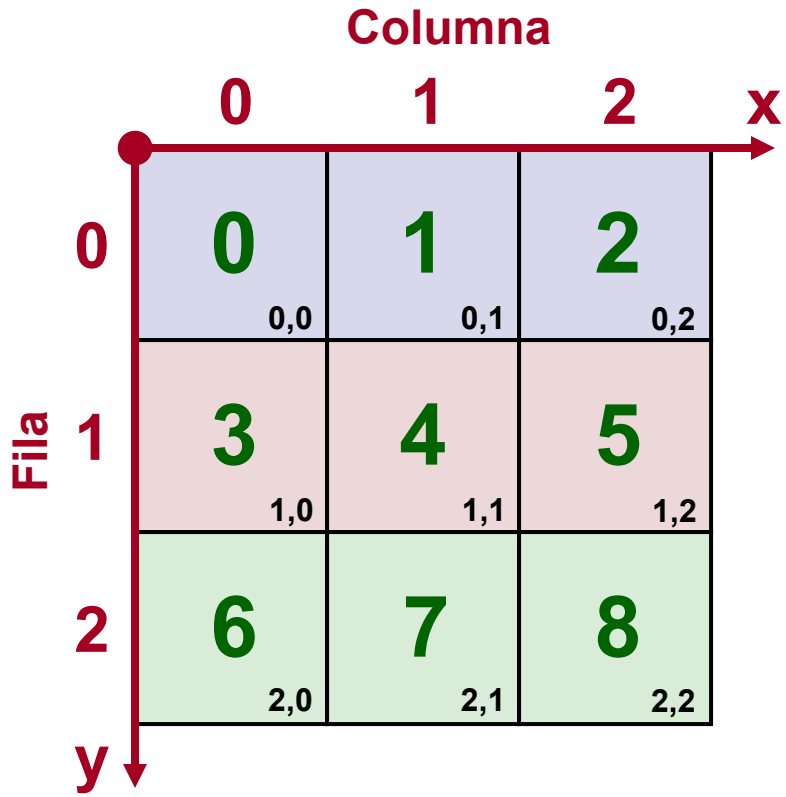
```
a[0][0] = 0;
a[0][1] = 1;
a[0][2] = 2;
```

fila 1

```
a[1][0] = 3;
a[1][1] = 4;
a[1][2] = 5;
```

fila 2

```
a[2][0] = 6;
a[2][1] = 7;
a[2][2] = 8;
```



Arrays

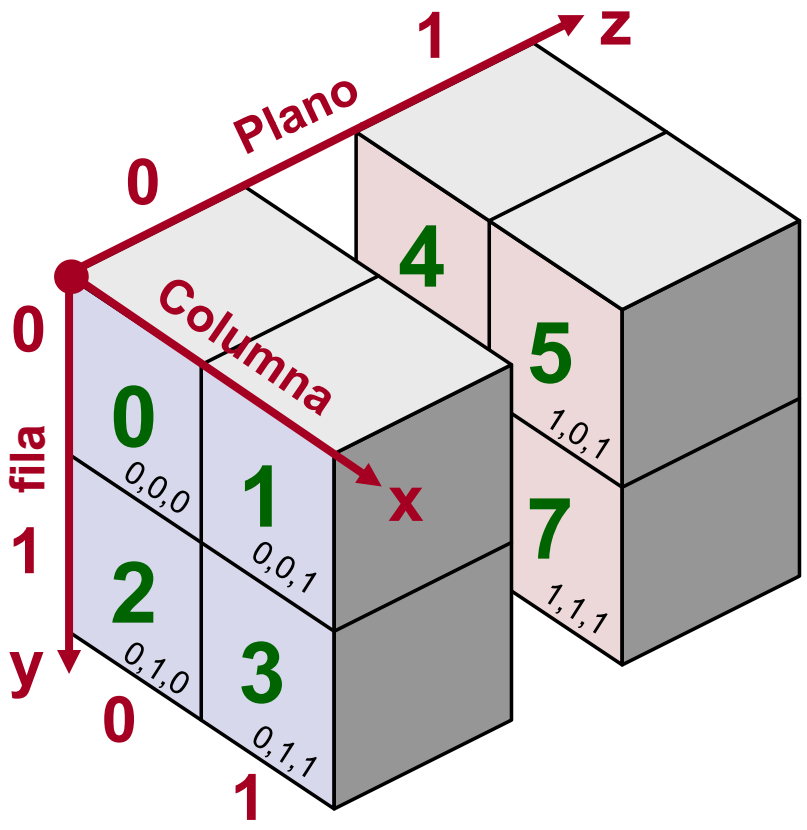
Visualizando Arrays de 3 dimensiones

```
int a[2][2][2] = {{{0, 1},{2, 3}},
                  {{4, 5},{6, 7}}};
```

Plano, Fila, Columna

a[z][y][x]

Plane 0	a[0][0][0] = 0;
	a[0][0][1] = 1;
	a[0][1][0] = 2;
	a[0][1][1] = 3;
Plane 1	a[1][0][0] = 4;
	a[1][0][1] = 5;
	a[1][1][0] = 6;
	a[1][1][1] = 7;



Arrays

Ejemplo de procesamiento de un array

```
/*****************************************
* Imprime 0 a 90 en incrementos de 10
*****************************************/
int main(void)
{
      int i = 0;
      int a[10] = {0,1,2,3,4,5,6,7,8,9};

      while (i < 10)
      {
            a[i] *= 10;
            printf("%d\n", a[i]);
            i++;
      }

      while (1);
}
```

Strings

Array de caracteres y Strings

Definición

Strings son array de **`char`** en los cuales el último caracter es un caracter nulo **`'\0'`** con un valor ASCII de 0. C no tiene tipos de datos string naticos s , los strings deben ser tratados como arrays de caracteres.

- **Strings:**
  - **Estan encerrados entre comillas `"string"`**
  - **Terminan con un caracter nulo `'\0'`**
  - **Deben ser tratados como una array de caracteres (tratados elemento por elemento)**
  - **Pueden ser inicializados por un string literal**

Strings

Creando un Array o String de Caracteres

Los Strings se crean como en cualquier otro array de `char`:

sintaxis

```
char nombrearray[longitud];
```

- ***longitud*** **debe ser un caracter mas que la longitud máxima del string a almacenarpara que tambien se pueda guardar el caracter el caracter nulo '\0'**
- **Un array `char` con n elementos contiene un string igual a n-1 `char`**

ejemplo

```
char str1[10];        //Contiene 9 caractere mas '\0'

char str2[6];         //contiene 6 caracteres mas '\0'
```

Strings

Como inicializar un String al Declararlo

Array de caracteres pueden ser inicializados con strings literales:

Sintaxis

```
char nombrearray[] = "Microchip";
```

- Array no requiere tamaño
- El tamaño es automaticamente determinado por la longitud de string
- caracter NULL `'\0'` es automaticamente adicionado

sintaxis

```
char str1[] = "Microchip"; //10 caracteres "Microchip\0"

char str2[5] = "Hola";     //5 caracteres "Hola\0"

//Declaración alternativa de string - requiere tamaño
char str3[4] = {'P', 'I', 'C', '\0'};
```

Strings

Como inicializar un string en Código

En código, puede inicializarse un string elemento por elemento:

Sintaxis

```
nombrearray[0] = char1;
nombrearray[1] = char2;
        :
nombrearray[n] = '\0';
```

- Caracter Null `'\0'` debe ser adicionado manualmente

Ejemplo

```
str[0] = 'H';
str[1] = 'o';
str[2] = 'l';
str[3] = 'a';
str[5] = '\0';
```

Strings

Comparando Strings

- Los Strings no pueden ser comparados por operadores relacionales (==, !=, etc.)
- Debe usar librería C Standar en funciones para manipulación de strings
- `strcmp()` retorna 0 si los string son iguales

Ejemplo

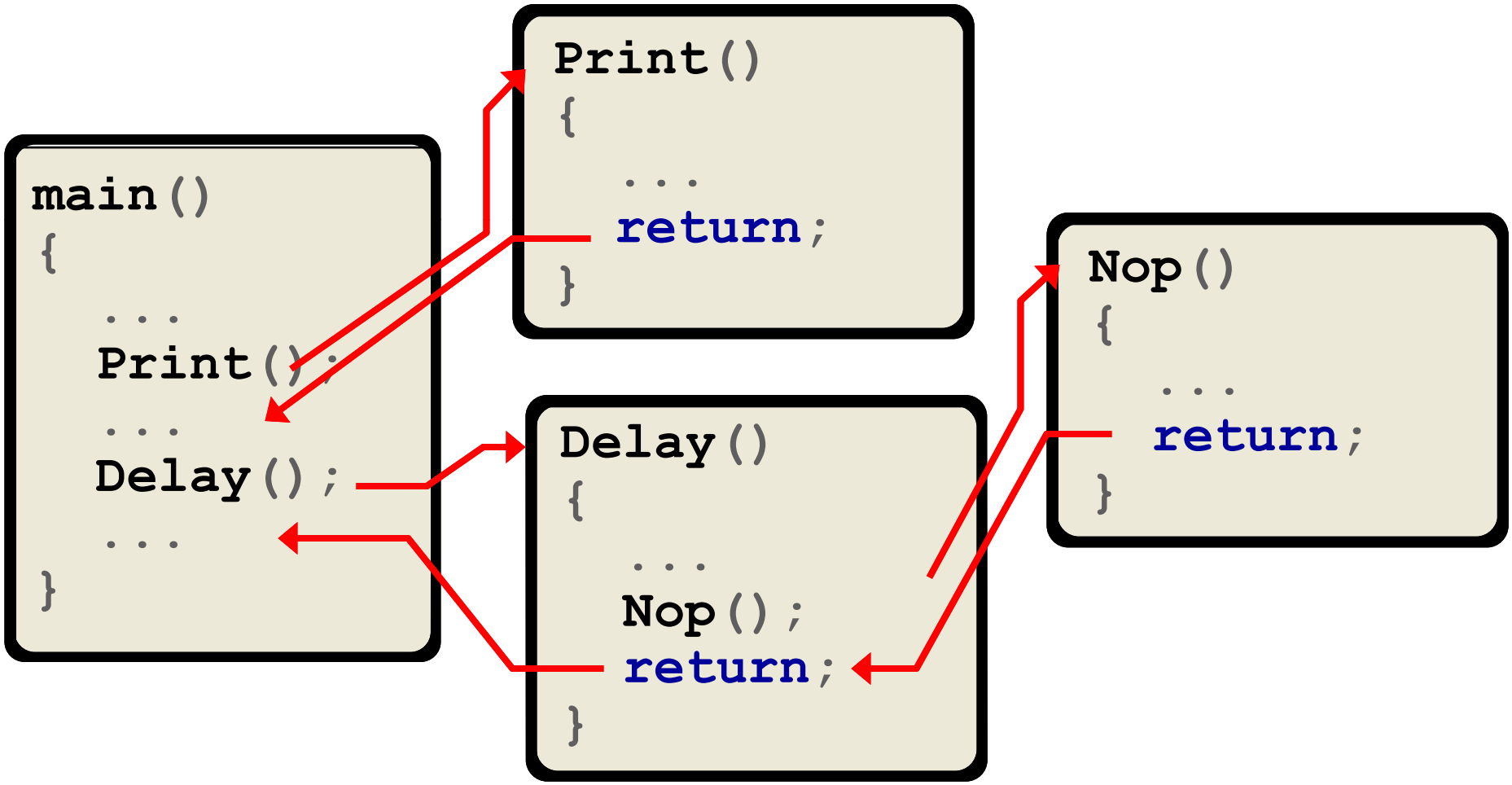
```
char str[] = "Hola";

if (!strcmp(str, "Hola"))
  printf("El string es \"%s\".\n", str);
```

Funciones

V3.2 August 10, 2007

Funciones

Estructura un programa



Funciones

Que es una función?

Definition

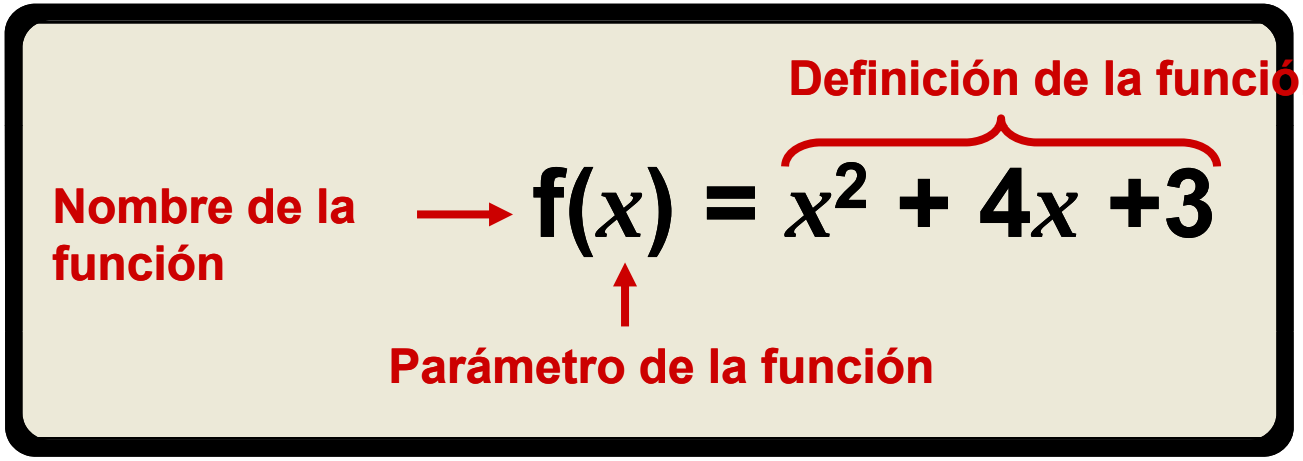
Funciones son segmentos de programa autónomos diseñados para realizar una tarea específica, bien definida.

- **Todos los programas en C tienen una o mas funciones**
- **La función `main()` es requerida**
- **Las funciones pueden tomar valores desde el código que las llama**
- **Las funciones retornan un valor simple**
- **Las funciones ayudan a organizar un programa en segmentos logicamente manejables**

Funciones

Recuerda las clases de Algebra?

- Las Funciones en C son conceptualmente son similares a las funciones matemáticas…



- Si usted pasa un valor 7 a la función: f(7), el valor 7 obtenido "copiado" dentro de x y usado donde quiera que x exisista con la función definida: f(7) = $7^2 + 4*7 + 3 = 80$

Funciones

Definiciones

Sintaxis

Tipo de dato retornado

expresion

Nombre

Parametro listado (opcional)

```
tipo identifier(tipo1 arg1,…,tipon argn)
{
  declaracón de sentencias
  return expresion;
}
```

Cuerpo

Cabecera

Retorna Evaluación (opcional)

Funciones

Definiciones de funciones: Retorna tipo de datos

sintaxis

```
tipo identificador(tipo1 arg1,…,tipon argn)
{
  declaraciones
  sentencias
  return expresion;
}
```

- El *tipo* de clarado de la función debe coincidir con el tipo de datos que *expresion* devuelva

Funciones

Definición de Funciones: Tipo de datos retornado

- **Una función puede tener declaraciones para retornos múltiples, pero solamente una será ejecutada y deben todas ser del mismo tipo**

ejemplo

```
int test(int a, int b)
{
      if (a > b)
            return 1;
      else
            return 0;
}
```

Funciones

Definición de Funciones: Tipo de datos retornados

- El tipo de función es `void` si:
  - La sentencia `return` no tiene *expresion*
  - La sentencia `return` no esta presente en todas
- Esto a veces llama a una función de procedimiento puesto que no se devuelve nada

Ejemplo

```
void identificador(tipo1 arg1,…,tipon argn)
{
  declaraciones
  sentencias
  return;
}
```

`return;` puede ser omitida si no se retorna nada

Funciones

Definición de Funciones: Parámetros

- Los parámetros de una función se declaran apenas como variables ordinarias, pero una coma delimita la lista dentro del paréntesis
- los nómbres de los parámetros solo son válidos dentro de la función (locales a la funcion)

Sintaxis

```
tipo identificador(tipo1 arg1,…,tipon argn)
{
  declaraciones
  sentencias
  return expresion;
}
```

Parámetros de funciones

Funciones

Definición de funciones: Parámetros

- Lista de parámetros puede mezclar datos
  - `int Txdata(int x, float y, char z)`
- Los parámetros del mismo tipo se deben declarar por separado - es decir:
  - `int maximum(int x, y)` no puede trabajar
  - `int maximum(int x, int y)` es correcta

Ejemplo

```
int maximum(int x, int y)
{
  return ((x >= y) ? x : y);
}
```

Funciones

Definición de funciones: Parámetros

- **Si no se requieren parámetros , usar la palabra clave `void` en lugar de la lista de parámetro al definir la función**

Ejemplo

```
tipo identificador(void)
{
      declaraciones
      sentencias
      return expresion;
}
```

Funciones

Como llamar o invocar una Funcion

Sintaxis de la llamada a función

- **Sin parámetros y no retorna valor**

```
Delay();
```

- **Sin parámetros pero con retorno de valor**

```
x = Delay();
```

- **Con parámetros pero sin retorno de valor**

```
Delay10KTCYx(a);
```

- **Con parámetros y retorna un valor**

```
x = itoa(a, b);
```

Funciones

Funcion Prototipos

- Justificada como variables, una función debe ser declarada antes de que pueda ser utilizada
- La declaración debe ocurrir antes de main () o de otras funciones que la utilicen
- Declaraciones pueden tener dos formas:
  - La definición de función entera
  - Justificada como funcion prototype – la definición de función en sí mismo se puede entonces poner dondequiera en el programa

Funciones

Funciones Prototipos

- Función prototipo puede tener dos formatos diferentes:
  - Una exacta copia de la función HEADER:

Ejemplo – Funcion Prototipo 1

```
int maximum(int x, int y);
```

  - Como la función main, pero sin los nombres de parámetro - solamente los tipos necesitan estar presentes para cada parámetro:

Ejemplo – Funcion Prototipo 2

```
int maximum(int, int);
```

Funciones

Declaracion y Uso: Ejemplo 1

Ejemplo 1

```
int a = 5, b = 10, c;

int maximo(int x, int y)
{
  return ((x >= y) ? x : y);
}

int main(void)
{
  c = maximo(a, b);
  printf("el maximo es: %d\n", c)
}
```

Funcion es *declarada* y *definida* antes de ser usada en el main()

Funciones

Declaracion y Uso: Ejemplo 2

Ejemplo 2

```
int a = 5, b = 10, c;

int maximo(int x, int y);

int main(void)
{
  c = maximo(a, b);
  printf("el maximo es %d\n", c)
}

int maximo(int x, int y)
{
  return ((x >= y) ? x : y);
}
```

Funcion es *declarada* con prototipo antes de ser usada en el main()

Funcion es *definida* despues de ser usada en el main()

Funciones

Pasaje de parámetros por Valor

- Parametros pasados a una función son pasados valor por valor
- Valores pasados a una función son copiados dentro de los parámetros de las variables locales
- La variable original que es pasada a la función no puede ser modificada por la funciónpuesto que solamente una copia de su valor fue pasada

Funciones

Pasaje de parámetros por Valor

Ejemplo

```
int a, b, c;

int contador(int x, int y)
{
  x = x + (++y);
  return x;
}

int main(void)
{
  a = 5;
  b = 10;
  c = contador(a, b);
}
```

El *valor* de `a` es *copiado* en `x`.
El *valor* de `b` es *copiado* en `y`.
La función no cambia el valor de `a` ó `b`.

Funciones y Alcance

Variables declaradas dentro de la función

- Las variables declararon dentro de un bloque de la función no son accesibles fuera de la función

Ejemplo

```
int x;
int acumula(int n)
{
      int a;
      return (a += n);
}
int main(void)
{
      x = acumula(5);
      x = a;
}
```

Esto puede generar un error. a no puede ser accedida fuera de la función donde ha sido declarada.

Funcione y Alcances

Variable Global versus Local

Ejemplo

```
int x = 5;

int suma(int y)
{
  int z = 1;
  return (x + y + z);
}

int main(void)
{
  int a = 2;
  x = suma(a);
  a = suma(x);
}
```

`x` puede ser vista por todos

`(suma)` parametro local es `y`
`(suma)` variable local es `z`

`(main)` variable local es `a`

Funciones y Alcance

Parameters

- ”Superposición" de nombres de variables:

n Declarada como Global y local

```
int n;

int acumula(int n)
{
      …
      y += n;
      …
}
```

n Declarada solo como global

```
int n;

int acumula(int x)
{
      …
      y += n;
      …
}
```

Funciones y Alcance

Parametros

Ejemplo

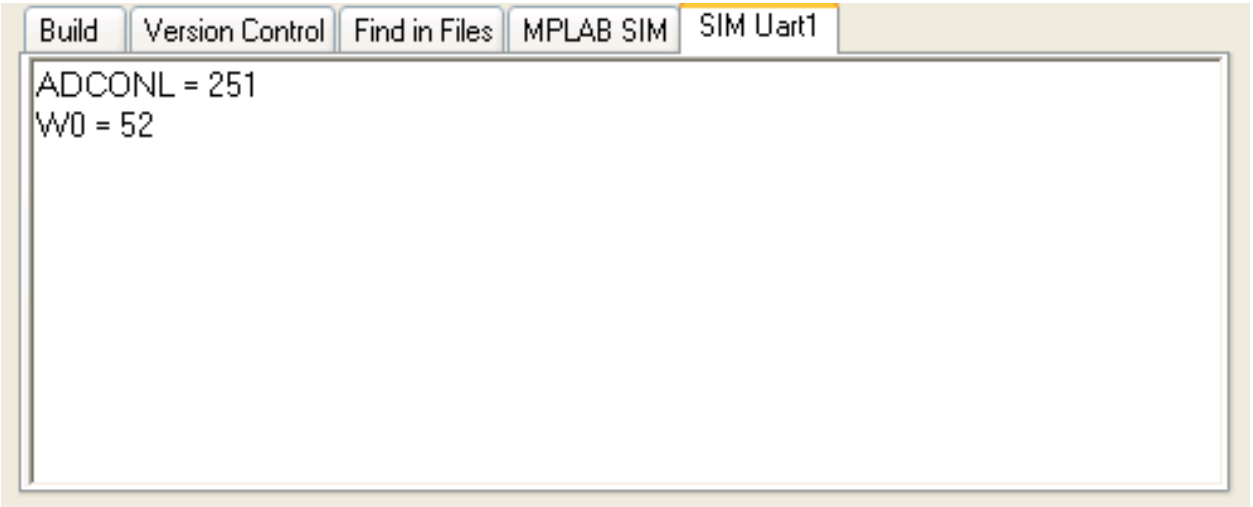
```
int n;

int acumula(int n)
{
      y += n;
}

int prod(int n)
{
      z *= n;
}
```

- Diferentes funciones pueden usar el mismo nombre del parámetro

printf()

Función de la Librería Standard

- Usada para escribir texto en la ”salida standard"
- Muchas en los microcontroladores PIC18 es la UART
- Esto nos permite depurar un programa ya sea usando el programa hyperteminal de Windows o en la USART virtual del MPLABSIM



printf()

Función de la Librería Standard

Sintaxis

```
printf(ControlString, arg1,…argn);
```

- Imprime todo palabra por palabra dentro del string excepto %d's los cuales son reemplazados por los valores de los argumentos del listado

Ejemplo

```
int a = 5, b = 10;
printf("a = %d\nb = %d\n", a, b);
```

Result:

```
a = 5
b = 10
```

NOTA: el 'd' en %d es la *conversion del caracter*. (ver la siguiente lamina para mas detalles)

printf()

Conversion Caracteres para Control de String

Conversion de caracter	Significado
%c	Caracter simple
%s	String (todo los caracteres hasta '\0')
%d	Entero sin signo
%o	Entero octal sin signo
%u	Entero decimal sin signo
%x	Entero decimal sin signo con digitos minuscula(1a5e)
%X	como x, pero con dígitos mayusula (e.g. 1A5E)
%f	Valor decimal con signo(floating point)
%e	Valor decimal con exponente(e.g. 1.26e-5)
%E	como e, pero usando E para el exponente (e.g. 1.26E-5)
%g	como e o f, pero dependiendo del tamaño y precision del valor
%G	como g, pero usando E para el exponente

printf()

- El valor mostrado es interpretado enteramente por el formato del string:

```
printf("ASCII = %d", 'a');
```

permite sacar : `ASCII = 97`

Un string mas problemático :

```
printf("Value = %d", 6.02e23);
```

permite sacar: `Valor = 26366`

- Resultados incorrectos pueden ser mostrados si el tipo de formato no corresponde al tipo de formato verdadero del argumento

printf()

Useful Format String Examples for Debugging

- Imprimir valor hexadecimal 16-bit con un prefijo "0x" y rellenar con ceros si es necesario para completar 4 dígtos hexa :

```
printf("direccion de x = %#06x\n", x_ptr);
```

#	Especifica que un 0x o 0X debería preceder un valor hexadecimal (tiene otro significado para diferentes conversiones de caracteres)
06	Especifica que 6 caracteres deben salir (incluido el prefijo 0x), los ceros permiten rellenar a la izquierda si es necesario
x	Especifica que el valor de salida debe ser expresado como un entero hexadecimal

printf()

Useful Format String Examples for Debugging

- **Como el anterior, excepto fuerza las letras hex mayusculas mientras deja 'x' en '0x' minusculas:**

```
printf("Dirección de x = 0x%04X\n", x_ptr);
```

`04` **Especifica que 4 caracteres deben ser sacados (no incluyendo 0x ya que se incluyo explicitamente en el string), los ceros seran completados en la izquierda en caso de necesidad**

`X` **Especifica que el valor debería ser expresado como un entero hexadecimal con A-F mayúsculas**

Directivas elementales

V3.2 August 10, 2007

Directivas

- Las directivas son ordenes que le damos al compilador, no son instrucciones de lenguaje C18.
- Existen varias directivas para el compilador, sin embargo las mas usadas son 3
  - #include: nos permite incluir un archivo
  - #define: permite redefinir una etiqueta
  - #pragma: se usa para definiciones de los fusibles de configuración y las interrupciones

Directiva `#include`

- Tres formas usan esta directiva `#include`:

Syntax

`#include <archivo.h>`

En este caso el compilador busca el archivo en el directorio actual y en el resto de los directorios

Por ejemplo: C:\Program Files\Microchip\

`#include "archivo.h"`

busca solo en el directorio actual

`#include "c:\Miproyecto\archivo.h"`

Usa un path específico para buscar un archivo include

Directiva #include

archivo HEADER main.h y achivo FUENTE main.c

main.h

```
unsigned int a;
unsigned int b;
unsigned int c;
```

El contenido del archivo main.h son efectivamente pasados dentro del comienzo del main.c en las lineas de directivas #include

main.c

```
#include "main.h"

int main(void)
{
        a = 5;
        b = 2;
        c = a+b;
}
```

Librerías de C18

V3.2 August 10, 2007

Las librerías de Microchip

- Microchip ha desarrollado una serie de librerías mediante las cuales es posible controlar todos los periféricos del microcontrolador, periféricos externos y hacer conversiones de datos
- Las librerías viene dentro del MCC18.
- Microchip documenta sus librerías profusamente en su manual :
  - MPLAB® C18 C COMPILER LIBRARIES (documento :DS51297F)

Las librerías de Microchip

- Las librerías se encuentran separadas por:
  - Librerías de periféricos de hardware
  - Librerías de periféricos por software
  - Librerías Generales por software
  - Librerías Matemáticas

Librerías para el control de Periféricos por Hardware

- Estas librerías permiten configurar y controlar los periféricos que incorporan los MCU PIC18:
  - Funciones I/O
  - Conversor A/D
  - Timers
  - Modulo Captura
  - PWM
  - I2C
  - SPI
  - USART
  - MicroWire

Librerías para el control de Periféricos por Software

- Estas librerías permiten configurar y controlar los siguiente periféricos
  - LCDs Inteligentes
    - Conexión en 4 u 8 bits
  - CAN 2510
  - I2C
  - SPI
  - USART

Librerías Generales de software

- Estas librerías permiten en tratamiento de caracteres, estas se clasifican en
  - Funciones para clasificación de caracteres
  - Funciones para la conversión de datos
  - Funciones para el manejo de Strings y memoria
  - Funciones de Delays
  - Funciones de Reset
  - Funciones para salida de caracteres

Funciones Matemáticas

- Estas librerías nos permiten procesar diversas funciones matemáticas como ser:
  - Funciones trigonométricas
  - Funciones exponenciales
  - Funciones logarítmicas

Librería para el control de un LCD

- **El MPLAB-C18 incorpora una serie de librerías para el control de periféricos por software. Una de ellas es la que controla un LCD inteligente.**
- **BusyXLCD :** esta ocupado el controlador del LCD ?
- **OpenXLCD :** Configura las lineas I/O usedas para controlar el LCD e inicializarlo.
- **putcXLCD :** Escribe un byte al LCD.
- **putsXLCD :** Escribe un string al LCD.
- **putrsXLCD :** Escribe un string desde memoria de programa del LCD.
- **ReadAddrXLCD:** Lee la dirección de un byte desde el LCD
- **ReadDataXLCD:** Lee un byte desde el LCD.

Librería para el control de un LCD

- **SetCGRamAddr :** Setea la dirección del generador de caracteres del LCD.
- **SetDDRamAddr :** Setea la dirección de datos del LCD.
- **WriteCmdXLCD :** Escribe un comando al LCD .
- **WriteDataXLCD** : Escribe un byte al LCD.

Ejemplo de aplicación del uso de las librerias LCD

- Para usar la librería XLCD el usuario debe editar el archivo xlcd.h para adaptar las conexiones del hardware de su aplicación por ejemplo:

```
23      /* Interface type 8-bit or 4-bit
24       * For 8-bit operation uncomment the #define BIT8
25       */
26       #define BIT8
27
28      /* When in 4-bit interface define if the data is in the upper
29       * or lower nibble.  For lower nibble, comment the #define UPPER
30       */
31      /* #define UPPER */
32
33      /* DATA_PORT defines the port to which the LCD data lines are connected */
34      #define DATA_PORT      PORTD
35      #define TRIS_DATA_PORT TRISD
36
37      /* CTRL_PORT defines the port where the control lines are connected.
38       * These are just samples, change to match your application.
39       */
40      #define RW_PIN   PORTEbits.RE0   /* PORT for RW */
41      #define TRIS_RW  DDREbits.RE0    /* TRIS for RW */
42      #define RS_PIN   PORTEbits.RE1   /* PORT for RS */
43      #define TRIS_RS  DDREbits.RE1    /* TRIS for RS */
44      #define E_PIN    PORTEbits.RE2   /* PORT for E  */
45      #define TRIS_E   DDREbits.RE2    /* TRIS for E  */
46
```

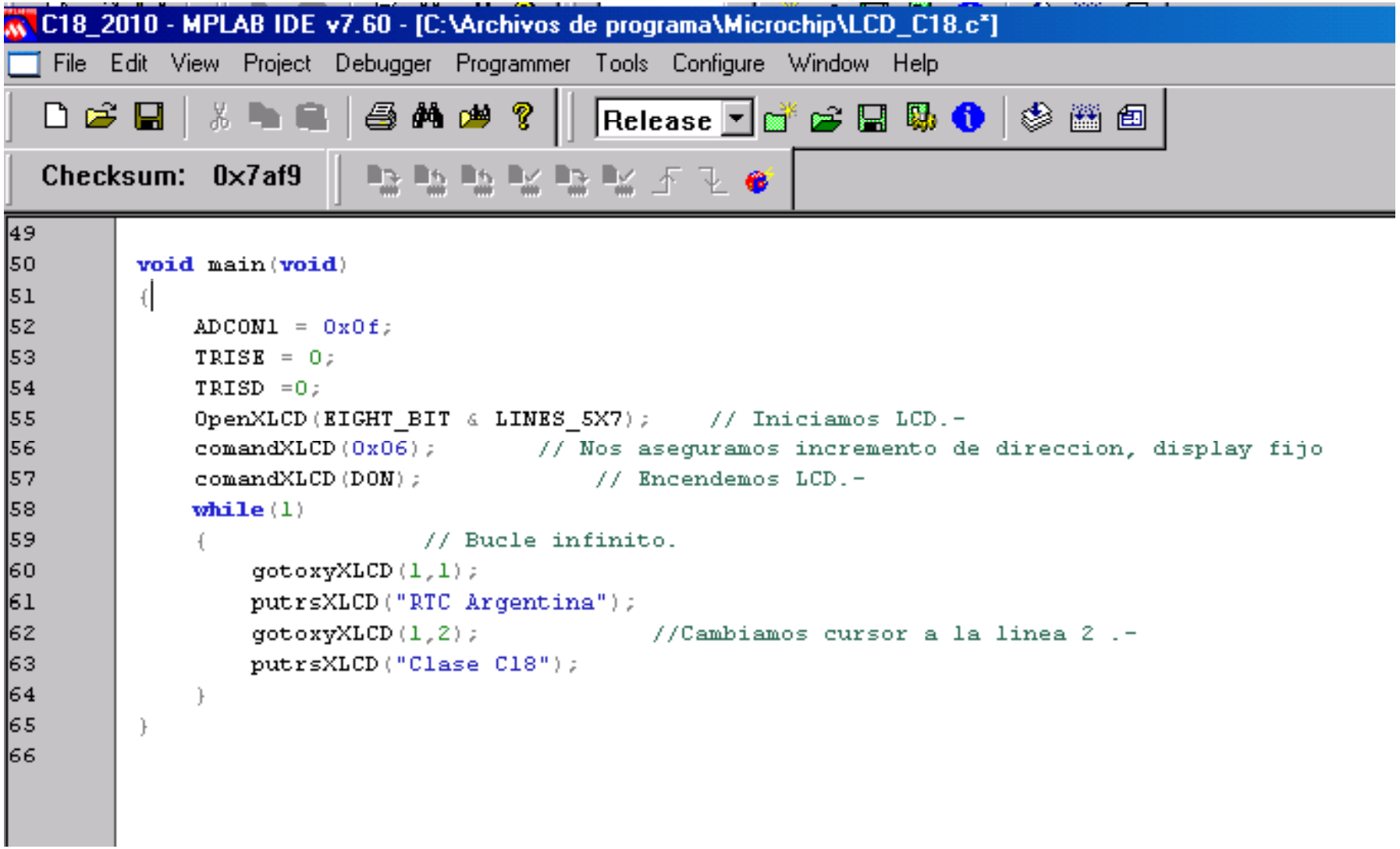
Programa LCD

```
1   #include <p18f4520.h>
2   #include <delays.h>
3   #include <xlcd.h>
4   #pragma config OSC = XT
5   #pragma config PWRT = ON
6   #pragma config WDT = OFF
7   #pragma config MCLRE = ON
8   #pragma config STVREN = OFF
9   #pragma config LVP = OFF
10
11  // Envia comando al LCD
12  void comandXLCD(unsigned char a){
13      BusyXLCD();
14      WriteCmdXLCD(a);
15  }
16  // Ubica cursor en (x = Posicion en linea, y = n° de linea)
17  void gotoxyXLCD(unsigned char x, unsigned char y){
18      unsigned char direccion;
19
20      if(y != 1)
21          direccion = 0x40;
22      else
23          direccion=0;
24
25      direccion += x-1;
26      comandXLCD(0x80 | direccion);
27  }
28  //
```

Programa LCD cont

```
//
/*****/
void DelayFor18TCY(void)
    {
        Delay100TCYx(0x2); //delays 20 cycles
        return;
    }
/*****/
void DelayPORXLCD(void)    // minimum 15ms
    {
        Delay100TCYx(0xA0);    // 100TCY * 160
        return;
    }
/*****/
void DelayXLCD(void)      // minimum 5ms
    {
        Delay100TCYx(0x36);       // 100TCY * 54
        return;
    }
/*****/
```

Programa LCD cont.




```
49
50      void main(void)
51      {
52          ADCON1 = 0x0f;
53          TRISE = 0;
54          TRISD =0;
55          OpenXLCD(EIGHT_BIT & LINES_5X7);     // Iniciamos LCD.-
56          comandXLCD(0x06);        // Nos aseguramos incremento de direccion, display fijo
57          comandXLCD(DON);             // Encendemos LCD.-
58          while(1)
59          {                // Bucle infinito.
60              gotoxyXLCD(1,1);
61              putrsXLCD("RTC Argentina");
62              gotoxyXLCD(1,2);             //Cambiamos cursor a la linea 2 .-
63              putrsXLCD("Clase C18");
64          }
65      }
66
```

C18 ADC Funciones de Librería

- **`#include <adc.h>`**
- **`char BusyADC (void);`**
  - Checquea si ADC esta esta procesando una conversión. Retorna ‘1’ si esta ocupado.
- **`void CloseADC (void);`**
  - Desabilita el ADC y ADC interrución
- **`void ConvertADC (void);`**
  - Inicia la conversión ADC
- **`void OpenADC (unsigned char config,`**
  `unsigned char config2);`
  - Usado para configurar el ADC
  - Algunos PIC18 Tiene un 3rd argumento **`portconfig`**
  - Valores de **`config`** y **`config2`** puede variar por dispositivo

C18 ADC Funciones de Librería

- **`int ReadADC (void);`**
  - Lea el resultado del conversor ADC
  - Retorna un valor 16-bit con signo
  - Resultado puede estar justificado a la derecha o a la izquierda determinado por la configuración
- **`void SetChanADC (unsigned char channel);`**
  - Selecciona el pin usado para entrada del ADC
  - El valor del puede ser ADC_CH0 hasta ADC_CH15)

Ejemplo de uso de las Librerias ADC.h

```
16      // Abre el Conversor AD
17      void OpenADC4520(void){
18          ADCON1=0b00001110;
19          ADCON2=0b10111000;
20          ADCON0= 0;
21          ADCON0bits.ADON = 1;
22          Delay1KTCYx( 5 ); // Delay for 5KTCYx
23          ADCON0bits.GO=1;
24          }
25
26      // Espera final de conversion
27      void BusyADC4520(void){
28          while (!(ADCON0bits.GO==1)){}
29          }
```

Ejemplo de uso de las Librerias ADC.h cont.

```
44      void main(void){
45      char mybuff [20];
46      unsigned int Canal0, Canal1;
47      char String[5];
48          OpenXLCD(EIGTH_BIT & LINES_5X7);     // Iniciamos LCD.-
49          comandXLCD(0x06);              // Nos aseguramos incremento de direccion, display fijo
50          comandXLCD(0x0C);              // Encendemos LCD.-
51          gotoxyXLCD(1,1);               // Ponemos el cursor en la linea 1
52          putrsXLCD("RTC Argentina"); // Escribimos el mensaje
53          gotoxyXLCD(1,2);               //Cambiamos cursor a la linea 2 .-
54          putrsXLCD("Clase C18");        // Escribimos el mensaje
55          Delay1KTCYx(500);
56          while(1)
57          {
58              OpenADC4520();
59              comandXLCD(0X01);
60              Delay1KTCYx( 5 ); // Delay for 5KTCYx
61              putrsXLCD("Conversion ADC");
62              Canal0= ReadADC();  // Realizamos la lectura.-
63              CloseADC(); // Disable A/D converter
64              gotoxyXLCD(1,2);               //Cambiamos cursor a la linea 2 .-
65              putrsXLCD("Ch0 = ");
66              itoa(Canal0, String);          // Convertimos entero a string.-
67              putsXLCD(String);
68              Delay1KTCYx(300);
69          }
70      }
```

Librerias para el Manejo de los Timers (Timers.h)

- Estas librerias nos permiten manejar el Timer0, Timer1, Timer2 , Timer3 y Timer4.
- Esta formada por las siguientes funciones:
  - OpenTimerX: permite configurar el Timer
  - CloseTimerX: apaga el Timer
  - WriteTimerX: nos permite escribir en el Timer
  - ReadTimerX: nos permite leer el Timer

  Nota: la X debe ser reemplazada por el Nro del timer que estemos trabajando

Librerias para el Manejo de los Timers (Timers.h)

- OpenTimer : nos permite configurar el funcionamiento del Timer.
- Según el Timer que seleccionemos sera el paso de parámetros:

Ejemplo:

- OpenTimer0(TIMER_INT_OFF &T0_8BIT & T0_SOURCE_INT & T0_PS_1_32);
- OpenTimer1(TIMER_INT_ON & T1_8BIT_RW & T1_SOURCE_EXT & T1_PS_1_1 & T1_OSC1EN_OFF & T1_SYNC_EXT_OFF);

Librerias para el Manejo de los Timers (Timers.h)

- ReadTimer: nos permite leer el valor de cuenta del Timer.
  - Ejemplo:

  timer0 = (unsigned char) ReadTimer0();
- WriteTimer: nos permite escribir un valor dentro del Timer
  - Ejemplo:

  WriteTimer0(10000);
- CloseTimer: desabilita el timer y desactiva la interrupción del mismo
  - Ejemplo:

  CloseTimer1();

Ejemplo de uso de la Librería Timers.h

```
void main(void){
unsigned int Timer=0;
char String[7];
    OpenXLCD(FOUR_BIT & LINES_5X7); // Iniciamos LCD.-
    OpenTimer1( TIMER_INT_OFF & T1_16BIT_RW & T1_SOURCE_EXT & T1_PS_1_8 & T1_OSC1EN_ON & T1_SYNC_EXT_O
    comandXLCD(0x06);           // Nos aseguramos incremento de direccion, display fijo
    comandXLCD(0x0C);           // Encendemos LCD.-
    while(1)
    {
    comandXLCD(0X01);
    Delay1KTCYx( 5 ); // Delay for 5KTCYx
    putrsXLCD("Timer 1=");
    Timer = ReadTimer1();    // Realizamos la lectura.-
    gotoxyXLCD(1,2);            //Cambiamos cursor a la linea 2 .-
    putrsXLCD("valor = ");
    itoa(Timer, String);        // Convertimos entero a string.-
    putsXLCD(String);
    Delay1KTCYx(50);
    }
}
```

Librerías para el manejo de la USART

- La librerias para el manejo de la USART nos permiten controlar la configuración, transmisión y recepción de datos:
  - BusyUSART: nos permite saber si ya se transmitió un dato
  - CloseUSART: desactivamos la USART
  - DataRdyUSART: nos permite saber si hay un dato disponible en el buffer de lectura de la USART
  - getcUSART: lee un byte desde la USART
  - getsUSART: lee un STRING desde la USART
  - OpenUSART: nos permite configurar la USART
  - putcUSART: escribimos un byte a la USART
  - putsUSART: escribimos un string desde la RAM a la USART
  - putrsUSART: escribimos un string desde la memoria de programa a la USART
  - ReadUSART: leemos un byte desde la USART
  - WriteUSART:escribimos un byte en la USART
  - baudUSART: seteamos el BAUD RATE en la USART mejorada

Programa ejemplo del uso de la librería de la USART cont.

```
void main(void){
char mybuff [20];
unsigned int Canal0, Canal1;
char String[5];
    OpenXLCD(FOUR_BIT & LINES_5X7); // Iniciamos LCD.-
    OpenUSART(USART_TX_INT_OFF & USART_RX_INT_ON & USART_ASYNCH_MODE & USART_EIGHT_BIT & USART
    comandXLCD(0x06);           // Nos aseguramos incremento de direccion, display fijo
    comandXLCD(0x0C);           // Encendemos LCD.-
    gotoxyXLCD(1,1);            // Ponemos el cursor en la linea 1
    putrsXLCD("RTC Argentina"); // Escribimos el mensaje en el LCD
    putrsUSART("RTC Argentina");// Esccribimos la en la USART
    putrsUSART("\r");
    putrsUSART("\n");
    gotoxyXLCD(1,2);            //Cambiamos cursor a la linea 2 .-
    putrsXLCD("Clase C18");     // Escribimos el mensaje
    putrsUSART("Clase C18");    // Esccribimos la en la USART
    putrsUSART("\r");
    putrsUSART("\n");
    Delay1KTCYx(500);
```

Programa ejemplo del uso de la librería de la USART cont.

```
    while(1)
    {
    OpenADC4520();
    comandXLCD(0X01);
    DelaylKTCYx( 5 ); // Delay for 5KTCYx
    putrsXLCD("Conversion ADC");
    putrsUSART("Conversión ADC");
    putrsUSART("\r");
    putrsUSART("\n");
    Canal0= ReadADC();  // Realizamos la lectura.-
    CloseADC(); // Disable A/D converter
    gotoxyXLCD(1,2);              //Cambiamos cursor a la linea 2 .-
    putrsXLCD("Ch0 = ");
    putrsUSART("Ch0 = ");
    itoa(Canal0, String);         // Convertimos entero a string.-
    putsXLCD(String);
    putsUSART(String);
    putrsUSART("\r");
    putrsUSART("\n");
    DelaylKTCYx(300);
    }
}
```

Laboratorios

V3.2 August 10, 2007

Donde conseguir ayuda

- Usted puede consultar:
- www.microchip.com

Fin

Gracias a todos!!!

V3.2 August 10, 2007