Estructura básica de un programa en Lenguaje C

Todos los programas (código fuente) de microcontroladores PIC en C tienen una estructura básica, a partir de la cual se desarrolla cualquier aplicación del usuario:

```
#Include <18f2520.h>           //Especificar el tipo de pic
#fuses intrc,nomclr,nobrownout //Definir fusibles
#use delay(internal=8M)        //Especificar la frecuencia del oscilador
Int1 j,k,am_pm=1;              //Declaración de variables de un solo bit.
Int x,y,w,m;                   //Declaración de var. enteras de 8 bits

void main( ){                  //Función principal
//Instrucciones del programa.
...
}
```

Los 7 elementos básicos de la programación de PIC en C

La programación de PIC en C se puede comprender mejor si se estudian sus elementos básicos; una vez que se dominen estos elementos se podrá dar solución a la gran mayoría de problemas de programación. El propósito de la mayoría de los programas es resolver un problema. Los programas resuelven los problemas por medio de la manipulación de información o datos. Normalmente los programas se caracterizan por permitir el ingreso de información, tener uno o varios lugares de almacenamiento de dicha información, contar con las instrucciones para manipular estos datos y obtener algún resultado del programa que sea útil para el usuario. También, las instrucciones se pueden organizar de tal forma que algunas de ellas se ejecuten sólo cuando una condición específica (o conjunto de condiciones) sea verdadera, otras instrucciones se repitan un cierto número de veces y otras pueden ser agrupadas en bloques que se ejecutan en diferentes partes de un programa.
Lo anterior constituye una breve descripción de los siete elementos básicos de la programación: **entrada de datos, tipos de datos, operaciones, salida, ejecución condicional, lazos y funciones**. Una vez que se dominan estos elementos se puede afirmar que se conocen los fundamentos de la programación, con lo cual ya es posible desarrollar una gran cantidad de aplicaciones de diversa índole.

Instrucciones básicas de PIC en C

1.- Instrucción de asignación (=)

Permite asignar a una variable un valor constante, el contenido de otra variable o el resultado de una expresión matemática. La asignación va de derecha a izquierda. Por ejemplo,

```
suma=0; //El valor 0 se almacena en la variable suma.

x0=x1; //El contenido de la variable x1 se almacena en la variable x0.
dx=(b-a)/n; //El resultado de la expresión matemática se almacena en la
variable dx.
```

2.- Instrucción de entrada de datos (variable=Input_B)
Permite el ingreso de uno o más datos a través de los pines del microcontrolador y almacenarlos en una o más variables. Por ejemplo;

```
x=input_a; //Los bits del puerto A se almacenan en la variable X.
```

También se puede leer el estado individual de cada bit de un puerto:

```
y = input(pin_b3) y= RB3_bit; //Lee el estado del pin RB3 y lo guarda en
la variable y.
```

3.- Instrucción de salida de datos (output_a)

Permite el envío de datos, el contenido de una variable o el resultado de una expresión matemática hacia los pines de un puerto. Por ejemplo,

```
Output_A(0x72); //En los pines del puerto A se muestra 0111,0010.
```

Como caso especial, se pueden enviar bits individuales a cada uno de los pines de un puerto:

```
Output_bit(pin_b0,0); //El pin RB0 se pone en 0.
```

4.- Instrucción de decisión (if...else)

Permite la ejecución de las instrucciones1 si la condición es verdadera, de lo contrario se ejecutan las instrucciones2. Las llaves { } no son necesarias cuando hay una sola instrucción.

```
if (condición){
 instrucciones1,2,3,…;
}
else{
 instrucciones 4,5,6….;
}
```

Ejemplo 1:

Si el contenido de la variable código es igual al contenido de la variable clave, se ejecutan las primeras cuatro instrucciones; de lo contrario se ejecutan únicamente los dos últimas instrucciones.

```
if (codigo==clave){
 intentos=0;
 output_high(pin_a7);
 delay_ms(1000);
 output_low(pin_a7);
}
 else{
 intentos++;
 Delay_ms(200);
}
```

Ejemplo 2:

Instrucción de decisión sin **else.** Es es una variante muy utilizada cuando se desea condicionar la ejecución de un grupo de instrucciones.

Las dos instrucciones se ejecutarán únicamente si la variable contador es igual a 2; de lo contrario la ejecución continúa a partir de la línea //Aquí.

```
if (contador==2){ //Sí la variable "contador" es igual a 2.
output_high(pin_a6);
contador=0;}
//aquí
```

Ejemplo **3:**

Similar al caso anterior pero con una sola instrucción. Si la variable horas es igual a 24 se reinicia esta variable con un valor de cero.

```
if (horas==24) horas=0;
```

Nota 1: Las condiciones se obtienen por medio de los operadores de relación y los operadores lógicos.

Nota 2: Operadores de relación:

```
>>   Mayor que
>=   Mayor o igual que
<<   Menor que
<=   Menor o igual que
==   Igual a (nótese la diferencia con
     el operador de asignación =)
!=   Distinto de
```

Nota 3: Operadores lógicos:

```
&&   Y
||   O
```

5.- Instrucción de ciclo controlado por una variable (for)
Permite ejecutar un grupo de instrucciones de manera repetitiva, una determinada cantidad de veces.

```
for (número de veces){
 instrucciones;
}
```

Ejemplo 1:
La variable i tiene un valor inicial de 7 (i=7) y un valor final de 1 (i>=1). Esta variable va disminuyendo de 1 en 1 (i--). Por lo tanto las dos instrucciones se ejecutarán en 7 ocasiones. La primera vez cuando i=7, la segunda cuando i=6, la

tercera cuando i=5 y así sucesivamente hasta la séptima vez cuando i=1. Luego la ejecución continúa a partir de la línea x=1;

```
for (i=7;i>=1;i--){//Desde i==7;mientras i>=1;decremente i
x=x<<1;            //corrimiento izquierdo de los bits de x una posición.
output_b(x);       //mostrar el valor actual de x
delay_ms(500);     //demora de 500 milisegundos
}
X=1;
```

Ejemplo **2:**

El valor inicial de la variable es 1 y su valor final es 3. La variable i se va incrementando de 1 en 1 (i++). Por lo tanto la instrucción se ejecuta tres veces, lo que da como resultado un retardo de 3 segundos. Luego la ejecución continúa a partir de la línea //Aquí.

```
for (i=1;i<=3;i++)  //asigna i=1;mientras i<=3;incremente i
Delay_ms(1000);     //demora de 1 segundo
//Aquí.
```

6.- Instrucción iterativa condicional (while)

Permite ejecutar un grupo de instrucciones de manera repetitiva, mientras una condición sea verdadera. Primero se revisa la condición para determinar su valor de verdad (verdadero o falso) y luego se ejecutan las instrucciones.

```
while (condición){
 instrucciones;
}
```

Ejemplo 1:

La ejecución del programa permanece indefinidamente en esta línea mientras que la terminal bo del puerto B sea igual a cero. Como caso particular no se ejecuta ninguna instrucción (la cual debería estar antes del punto y coma).

```
while (!input(pin_b0);)  //Leer el pin_b0 mientras que sea igual a cero;
```

Ejemplo de bucle infinito2: En lenguaje C, cualquier valor numérico diferente a cero se considera VERDADERO, y un valor numérico igual a cero se considera FALSO.
Al valor numérico de la variable x se le suma el valor 65, el resultado se envía hacia los pines del puerto B. Este proceso se repite continua e indefinidamente, debido a que la condición siempre es verdadera (1).

```
while (true){  //mientras sea verdadero
x=x+65;        //sume 65 a la variable x
output_b(x);}  //muestre el vaor actual de x por el Puerto B
```

Ejemplo 3:

Las cuatro instrucciones encerradas por { } se ejecutarán indefinidamente mientras el valor del bit B0 sea igual a 0.

```
while (!input(pin_b0);){ //Mientras el pin_b0 sea igual a 0;
output_high(pin_B1);     //Mostrar un nivel alto en el pin_b1;
delay_ms(500);           //Demora de 500 ms
output_low(pin_b1);      //Mostrar un nivel bajo en el pin_b1;
delay_ms(200);           //Demora de 200 mS
}
```

7.- Instrucción hacer-mientras (do...while)

Permite ejecutar un grupo de instrucciones de manera repetitiva, mientras una condición sea verdadera. Es similar a la instrucción while, con la diferencia de que primero se ejecutan las instrucciones y luego se revisa la condición.

```
do{
 instrucciones;
}
while (condición);
```

Ejemplo 1:

La variable c tiene un valor inicial de 255. La instrucción c=getc(); se ejecuta y luego se revisa la condición (c==255). Mientras c sea igual a 255 la condición será VERDADERA . Como resultado se tendrá un lazo infinito mientras la variable c tenga el valor de 255. Cuando la variable c cambie de valor como consecuencia del valor pulsado por una tecla, la condición será FALSA y la ejecución continuará en la línea siguiente al while (//Aquí).

```
c=255; //Asignar el valor de 255 a la variable C;
do {   //Realizar las operaciones encerradas entre{}
c=getc(); //Asignar a c el valor capturado por el RS232c
}
while (c==255);  /*Repetir las instrucciones encerradas entre
los símbolos {} mientras c sea igual a 255 */
//Aquí.
```

Ejemplo 2:

Las tres instrucciones dentro de { } se ejecutarán indefinidamente mientras la variable “tecla” sea diferente a 1.

```
do{            /*Realizar las operaciones encerradas entre {}
 pp =input(b); //asignar a la variable pp el valor del Puerto B
 result=sqrt(pp); //Asignar a la variable result la raíz cuadrada de pp
 output_c(result);//Mostrar por el Puerto c la variable result
 }
while (tecla!= 1); Mientras que la variable tecla sea diferente de 1
```

Nota: A diferencia de la instrucción while, en la instrucción do...while las instrucciones se ejecutan por lo menos una vez.

8.- Instrucción de selección múltiple (switch).

Permite la ejecución de un grupo de instrucciones de varios grupos posibles, dependiendo del valor de una variable.

```
switch (variable){
 case 1: instrucciones del grupo 1;
         break;
 case 2: instrucciones del grupo 2;
         break;
 case 3: instrucciones del grupo 3;
         break;
 default: instrucciones;
}
```

Si la variable es igual a 1 se ejecutan únicamente las instrucciones del grupo 1, si es igual a 2 se ejecutan únicamente las instrucciones2 y así sucesivamente. Si la variable no es igual a ninguno de los casos (case) se ejecutan las instrucciones por defecto (default).

Ejemplo 1:

Esta es una función numérica que da como resultado un número hexadecimal dependiendo del valor que tenga la variable digit0. Si digit0 es igual a 0 la función muestra por el puerto b el valor 0x40. Si digito es igual a 1, la función muestra 0x79, y así sucesivamente. Este ejemplo es una variante de la instrucción switch, donde no aparece el elemento default.

```
switch (digit){
  case 0: output_b(0x40);
  break;
  case 1: output_b(0x79);
  break;
  case 2: output_b(0x24);
  break;
  case 3: output_b(0x30);
  break;
  case 4: output_b(0x19);
  break;
  case 5: output_b(0x12);
  break;
  case 6: output_b(0x02);
  break;
  case 7: output_b(0x78);
  break;
  case 8: output_b(0x00);
  break;
  case 9: output_b(0x10);
  break; }
}
```

Funciones
Una función es una agrupación de instrucciones para formar una nueva instrucción creada por el programador (usuario). Empleando funciones, la solución total de un determinado problema se divide en varios sub-problemas, cada uno de los cuales es resuelto por medio de una función particular, aplicando de esta manera la conocida máxima “Divide y vencerás”.

Las funciones constituyen una de las características fundamentales de lenguaje C, pues todo programa bien escrito hace uso de ellas. Para poder utilizar una función se deben cumplir los dos siguientes pasos:

1.-Declaración de la función.- Consiste en indicar el tipo, nombre y parámetros de la función:

```
tipo nombre ( parámetro1, parámetro2, ...)
//Ejemplo
#int_ext   //Declara que la función a continuación es un interrupción.
void ext_e0()  //nombre de la función(no tiene parámetros)
```

2.-Definición de la función.- Consiste en indicar las instrucciones que forman parte de dicha función:

```
void ext_e0(){     //tipo nombre (sin parámetros)
x++;
if((x==12)&&(y==0)){output_toggle(pin_a2);
output_toggle(pin_a1);
if(x>=13){x=1;}
}

tipo nombre ( parámetro1, parámetro2, ...){
instrucciones;
}
```

Estructura básica de un programa en lenguaje C (con funciones)
Todos los programas (código fuente) en lenguaje C tienen una estructura básica, a partir de la cual se desarrolla cualquier aplicación del usuario:

```
//incluir el pic a utilizar
//definir fusibles
//especificar la frecuencia del oscilador
//incluir el dispositivos asociados
//declarar variables globales.
//**Declaración de funciones (prototipos)**...
//******Fin de declaración de funciones*****
//===Declaración de variables de la función===

//**Función principal***
```

```
void main( ){
//Instrucciones del programa.
}
========Definición de funciones================

función1{
 instrucciones1;
}

función2{
 instrucciones2;
}
//============================Fin de definición de funciones==========================
```

Nota 1: Los tipos de funciones más empleadas son numéricas (char) y nulas (void). Las primeras retornan (return) o devuelven como resultado un número, mientras que las segundas simplemente ejecutan un grupo de instrucciones.

Ejemplo 1:

Este ejemplo se puede encontrar en el capítulo II, Puertos Digitales, del libro. La función es numérica (char), su nombre es Bin2_7seg y tiene un parámetro digit de tipo char.

La función se utiliza como si fuera una instrucción cualquiera, tomando en cuenta el tipo de función y su(s) parámetro(s). En este ejemplo se tiene PORTB=Bin2_7seg(PORTA). Esto significa que la variable PORTA remplaza a la variable digit. Por lo tanto si PORTA es igual a 0, la función devuelve el valor 0x3F que será enviado al puerto B. Si PORTA es igual a 1, la función devuelve 0x06 que será enviado al puerto B, y así sucesivamente.

```
//7seg1.c
//Se utiliza la función Bin2_7seg que transforma un número binario a su
//equivalente en 7 segmentos.

char Bin2_7seg(char digit); //Prototipo de la función.

void main(){
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFS==0);//Esperar mientras el oscilador está inestable.
PORTA=0x00;  //Inicialización.
PORTB=0x00;
ANSEL=0x00;  //Pines AN<6:0> como E/S digital.
TRISB=0x00;  //Puerto B como salida.
```

```
while (1) PORTB=Bin2_7seg(PORTA);
}

char Bin2_7seg(char digit){  //Definición de la función.
 switch (digit){
   case 0: return 0x3F;  //0x3F es el código 7-segmentos del
0.
   case 1: return 0x06;  //0x06 es el código 7-segmentos del
1.
   case 2: return 0x5B;
   case 3: return 0x4F;
   case 4: return 0x66;
   case 5: return 0x6D;
   case 6: return 0x7D;
   case 7: return 0x07;
   case 8: return 0x7F;
   case 9: return 0x67;
 }
}
```

Ejemplo 2:

Variante del ejemplo anterior, en el que se hace únicamente la definición de la función (sin declaración). Se debe hacer antes de la función principal, de lo contrario se producirán errores de compilación por tratar de usar una función desconocida.

```
//7seg1.c
//Se utiliza la función Bin2_7seg que transforma un número
binario a su
//equivalente en 7 segmentos.

char Bin2_7seg(char digit){  //Definición de la función.
 switch (digit){
   case 0: return 0x3F;  //0x3F código 7-segmentos del 0.
   case 1: return 0x06;  //0x06 código 7-segmentos del 1.
   case 2: return 0x5B;   //0x5b código 7-segmentos del 2.
   case 3: return 0x4F;
   case 4: return 0x66;
   case 5: return 0x6D;
   case 6: return 0x7D;
   case 7: return 0x07;
   case 8: return 0x7F;
   case 9: return 0x67;
 }
}
```

```
void main(){
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFS==0);//Esperar mientras el oscilador está
inestable.
PORTA=0x00;  //Inicialización.
PORTB=0x00;
ANSEL=0x00;  //Pines AN<6:0> como E/S digital.
TRISB=0x00;  //Puerto B como salida.
while (1) PORTB=Bin2_7seg(PORTA);
}
```

Detalles importantes de Lenguaje en CCs para tener en cuenta
En la programación de PIC en C existen pequeños detalles que se deben tener muy en cuenta y que permitirán que los programas realizados cumplan las tareas para los cuales fueron diseñados. Con respecto a los comentarios, se puede decir que son importantes aunque no son necesarios. Su importancia radica en que ofrecen una mayor facilidad para entender el código y realizar su mantenimiento (esto es especialmente cierto cuando ha pasado el tiempo y necesitamos realizar alguna modificación).

- Los comentarios se inician con la doble barra diagonal //.
- Los signos de agrupación siempre deben estar en pareja, es decir si hay tres llaves de apertura {{{, deben haber tres llaves de cierre co }}}. Lo mismo con los paréntesis ( ).
- Los números hexadecimales se escriben comenzando siempre con 0x, por ejemplo 0x0A, 0x16, 0xFD, etc.
- Los números binarios se escriben comenzando siempre con 0b, por ejemplo 0b00001110; 0b11101111, etc.
- Los números decimales se escriben de la forma común y corriente, por ejemplo 64, 126, 12.75, etc.
- No se debe confundir el operador de asignación (=) con el operador de comparación (==) igual a.
- El punto y coma (;) indica el final de una instrucción, por lo tanto hay que tener mucho cuidado para colocarlo en el lugar apropiado.
- Las llaves { } no son necesarias en aquellos casos en los que únicamente se va a ejecutar una instrucción (ver los ejemplos a lo largo de este apartado).
- Todo programa en Lenguaje C debe tener una función principal (main), y su nombre no debe ser cambiado.

Los tipos de datos más utilizados se muestran en la tabla 1.5.

Tipo	Tamaño (en bytes)	Rango
bit	1 bit	0 ó 1
char	1	0…255
signed char	1	-128…127
int	2	-32768…32767
unsigned	2	0…65535
long	4	-2147483648…2147483647
unsigned long	4	0…4294967295
float	4	$-1.5 * 10^{45}$ … $+3.4 * 10^{38}$

Tabla 1.5 Tipos de datos más comunes en mikroC PRO

El tipo float es para números con punto decimal, mientras que los demás tipos son para números enteros.

La EJECUCIÓN DE UNA INSTRUCCIÓN consiste en la realización de las operaciones especificadas en esa instrucción. De la ejecución se encarga la CPU (unidad central de proceso) del microcontrolador PIC.