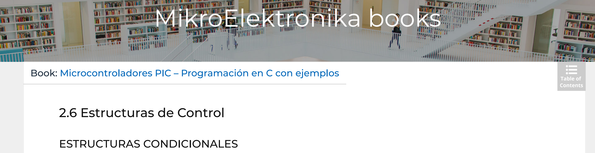

2.6 Estructuras de Control

ESTRUCTURAS CONDICIONALES

Las condiciones son ingredientes comunes de un programa. Las condiciones permiten ejecutar una o varias sentencias dependiendo de validez de una expresión. En otras palabras, 'Si se cumple la condición (…), se debe hacer (…). De lo contrario, si la condición no se cumple, se debe hacer (…)'. Los operandos condicionales **if-else** y **switch** se utilizan en las operaciones condicionales. Una sentencia condicional puede ser seguida por una sola sentencia o por un bloque de sentencias a ser ejecutadas.

OPERADOR CONDICIONAL if-else

El operador **if** se puede utilizar solo o asociado al operador **else (if-else)**. Ejemplo del operador **if**:

```
if(expresión) operación;
```

Si el resultado de la expresión encerrada entre paréntesis es verdadero (distinto de 0) la operación se realiza y el programa continúa con la ejecución. Si el resultado de la expresión es falso (0), la operación no se realiza y el programa continúa inmediatamente con la ejecución. Como hemos mencionado, la otra forma combina tanto el operador if como el else:

```
if(expresión) operación1 else operación2;
```

Si el resultado de la expresión es verdadero (distinto de 0), se realiza operación1, de lo contrario se realiza la operación2. Después de realizar una de las operaciones, el programa continúa con la ejecución. La sentencia **if-else** se parece a lo siguiente:

```
if(expresión)
operación1
else
operación2
```

Si operación1 u operación2 está compuesta, escriba una lista de sentencias encerradas entre llaves. Por ejemplo:

```
if(expresión) {
... //
... // operación1
...} //
else
operación2
```

El operador **if-else** se puede sustituir por el operador condicional '?:':

```
(expresión1)? expresión2 : expresión3
```

Si el valor de la expresión1 es distinto de 0 (verdadero), el resultado de la expresión entera será equivalente al resultado obtenido de laexpresión2. De lo contrario, si la expresión1 es 0 (falso), el resultado de la expresión entera será equivalente al resultado obtenido de laexpresión3. Por ejemplo:

```
maximum = (a>b)? a : b // A la variable maximum se le asigna el
                       // valor de la variable mayor(a o b)
```

Operador Switch

A diferencia de la sentencia **if-else** que selecciona entre dos opciones en el programa, el operador **switch** permite elegir entre varias opciones. La sintaxis de la sentencia **switch** es:

```
switch (selector) // Selector es de tipo char o int
{
  case constante1:
     operación1 // El grupo de operadores que se ejecutan si
     ...        // el selector y la constante1 son equivalentes

     break;

     case constante2:

     operación2 // El grupo de operadores se ejecuta si
     ...        // el selector y la constante2 son equivalentes

     break;
     ...
  default:

     operación_esperada // El grupo de operadores que se ejecuta si
     ...                // ninguna constante equivale al selector
     break;
}
```

La operación switch se ejecuta de la siguiente manera: primero se ejecuta el selector y se compara con la constante1. Si coinciden, las sentencias que pertenecen a ese bloque se ejecutan hasta llegar a la palabra clave break o hasta el final de la operación switch. Si no coinciden, el selector se compara con la constante2. Si coinciden, las sentencias que pertenecen a ese bloque se ejecutan hasta llegar a la palabra clave break etc. Si el selector no coincide con ninguna constante, se ejecutarán las operaciones que siguen al operador default. También es posible comparar una expresión con un grupo de constantes. Si coincide con alguna de ellas, se ejecutarán las operaciones apropiadas:

```
switch (días) // La variable días representa un día de la semana.
{ // Es necesario determinar si es un día laborable o no lo es
   case1:case2:case3:case4:case5: LCD_message = 'Día laborable'; break;
   case6:case7: LCD_message = 'Fin de semana'; break;
   default:LCD_message_1 = 'Elija un día de la semana'; break;
}
```

La palabra clave de C 'break' se puede utilizar en cualquier tipo de bloques. Al utilizar 'break', es posible salir de un bloque aunque la condición para su final no se haya cumplido. Se puede utilizar para terminar un bucle infinito, o para forzar un bucle a terminar antes de lo normal.

BUCLES

A menudo es necesario repetir una cierta operación un par de veces en el programa. Un conjunto de comandos que se repiten es denominado un bucle de programa. Cuántas veces se ejecutará, es decir cuánto tiempo el programa se quedará en el bucle, depende de las condiciones de salir del bucle.

Bucle While

El bucle while se parece a lo siguiente:

```
while(expresión){
comandos
...
}
```

Los comandos se ejecutan repetidamente (el programa se queda en el bucle) hasta que la expresión llegue a ser falsa. Si la expresión es falsa en la entrada del bucle, entonces el bucle no se ejecutará y el programa continuará desde el fin del bucle while. Un tipo especial del bucle de programa es un bucle infinito. Se forma si la condición sigue sin cambios dentro del bucle. La ejecución es simple en este caso ya que el resultado entre llaves es siempre verdadero (1=verdadero), lo que significa que el programa se queda en el mismo bucle:

```
while(1){ // En vez de "while(1)", se puede escribir "while(true)"
...       // Expresiones encerradas entre llaves se ejecutarán
...       // repetidamente (bucle infinito)
}
```

Bucle For

El bucle **for** se parece a lo siguiente:

```
for(expresión_inicial; expresión_de_condición; cambiar_expresión) {
   operaciones
   ...
}
```

La ejecución de esta secuencia de programa es similar al bucle while, salvo que en este caso el proceso de especificar el valor inicial (inicialización) se realice en la declaración. La expresión_ inicial especifica la variable inicial del bucle, que más tarde se comparan con la expresión_ de_condición antes de entrar al bucle. Las operaciones dentro del bucle se ejecutan repetidamente y después de cada iteración el valor de la expresión_inicial se incrementa de acuerdo con la regla cambiar_expresión. La iteración continúa hasta que la expresión_de_condición llegue a ser falsa.

```
for(k=0; k<5; k++) // La variable k se incrementa 5 veces (de 1 a 4) y
operación          // cada vez se repite la expresión operación
...
```

La operación se ejecutará cinco veces. Luego, al comprobar se valida que la expresión k<5 sea falsa (después de 5 iteraciones k=5) y el programa saldrá del bucle for.

Bucle Do-while

El bucle do-while se parece a lo siguiente:

```
do
operación
while (cambiar_condición);
```

La expresión cambiar_condición se ejecuta al final del bucle, que significa que operación se ejecuta como mínimo una vez sin reparar en que si la condición es verdadera o falsa. Si el resultado es distinto de 0 (verdadero), el procedimiento se repite. Todos los siguientes ejemplos son equivalentes. Esta parte del código visualiza "hello" en un LCD 10 veces con un retardo de un segundo. Note que en este ejemplo se utilizan funciones predefinidas, que se encuentran en las librerías del compilador mikroC PRO for PIC. No obstante le aconsejamos que no trate de entenderlas en detalle. Su comportamiento general dentro del bucle se explica por medio de los comentarios.

```
i = 0; // Inicialización del contador

while (i<10) {          // Condición
  Lcd_Out(1,3,"hello"); // Visualizar "hello" en el LCD
  Delay_ms(1000);       // Retardo de 1000 ms
  Lcd_Cmd(_LCD_CLEAR);  // Borrar el LCD
  Delay_ms(500);        // Retardo de 500ms
  i++;                  // Contador se incrementa
}
```

```
for(i=0; i<10; i++) {     // Inicialización, condición, incremento
   Lcd_Out(1,3,"hello");  // Visualizar "hello" en el LCD
   Delay_ms(1000);        // Retardo de 1000 ms
   Lcd_Cmd(_LCD_CLEAR);   // Borrar el LCD
   Delay_ms(500);         // Retardo de 500ms
}
```

```
i = 0;                    // Inicialización del contador
do {
  Lcd_Out(1,3,"hello");  // Visualizar "hello" en el LCD
  Delay_ms(1000);        // Retardo de 1000 ms
  Lcd_Cmd(_LCD_CLEAR);   // Borrar LCD
  Delay_ms(500);         // Retardo de 500ms
  i++;                   // Contador se incrementa
}
while (i<10);             // Condición
```

SENTENCIAS DE SALTO

SENTENCIA BREAK

A veces es necesario detener y salir de un bucle dentro de su cuerpo. La sentencia break se puede utilizar dentro de cualquier bucle (while, for, do while) y en las sentencias switch también. En éstas la sentencia break se utiliza para salir de las sentencias switch si la condición case es verdadera. En este ejemplo, "Esperar" está parpadeando en la pantalla LCD hasta que el programa detecte un uno lógico en el pin 0 del puerto PORTA.

```
while(1){ // Bucle infinito
  if(PORTA.F0 == 1)       // Probar si el estado lógico del pin 0 del puerto
  break;                  // PORTA es 1; si equivale, salir del bucle
  Lcd_Out(1,3,"Esperar"); // Visualizar "Esperar" en el LCD
  Delay_ms(1000);         // Retardo de 1000 ms
  Lcd_Cmd(_LCD_CLEAR);    // Borrar LCD
  Delay_ms(500);          // Retardo de 500ms
}
```

SENTENCIA CONTINUE

La sentencia continue colocada dentro de un bucle se utiliza para saltar una iteración. A diferencia de la sentencia break, el programa se queda dentro del bucle y las iteraciones continúan.

```
// Si x=7, puede ocurrir una división por 0,
// continue se utiliza aquí para evitar esta situación.
x=1;
while (x<=10) {
  if (x == 7) { // saltar x=7 para evitar división por 0
    Lcd_Cmd(_LCD_CLEAR);
    Lcd_Out(1,3,"División by 0");
    Delay_ms(1000);
    x++;
    continue; // Después de esta línea, saltar a la sentencia while con x=8
  }

  a = 1/(x-7); // Esta división generará un error si x=7

  /* Muchas operaciones pueden ocurrir aquí */

  Lcd_Out(1,3,"División is OK"); // Poner este mensaje en el LCD
  Delay_ms(1000);
  x++;
}
```

SENTENCIA GOTO

La sentencia goto le permite hacer un salto absoluto al otro punto en el programa. Esta característica se debe utilizar con precaución ya que su ejecución puede causar un salto incondicional sin hacer caso a todos los tipos de limitaciones de anidación. El punto destino es identificado por una etiqueta, utilizada como un argumento para la sentencia goto. Una etiqueta consiste en un identificador válido seguido por un colon (:).

```
...
if(CO2_sensor) goto aire acondicionado; // Si se consta que el valor
...                                     // de la variable CO2_sensor =1
                                        // hacer salto a la línea de programa
                                        // Aire acondicionado
...
Aire acondicionado:                     // Desde aquí sigue la parte del código que se ejecutará
                                        // en caso de una concentración de CO2 demasiado alta
...                                     // en el ambiente
```