# MikroElektronika books

Book: Microcontroladores PIC – Programación en C con ejemplos

Table of Contents

## 2.4 Variables y Constantes

### Definiciones

Una variable es un objeto nombrado capaz de contener un dato que puede ser modificado durante la ejecución de programa. En C, las variables tienen tipo, que significa que es necesario especificar el tipo de dato que se le asigna a una variable (*int*, float etc.). Las variables se almacenan en la memoria RAM y el espacio de memoria que ocupan (en bytes) depende de su tipo.

```
/* dos lineas de programa consecutivas. En la primera linea del programa
se define el tipo de variable */

int a = 1000; // Variable a es de tipo int y equivale a 1000
a = 15;       // a equivale a 15
```

Una constante tiene las mismas características que una variable excepto el hecho de que su valor asignado no puede ser cambiado durante la ejecución de programa. A diferencia de las variables, las constantes se almacenan en la memoria Flash del microcontrolador para guardar el mayor espacio posible de memoria RAM. El compilador las reconoce por el nombre y el prefijo const. En mikroC, el compilador reconoce automáticamente el tipo de dato de una constante, así que no es necesario especificar el tipo adicionalmente.

```
/* dos lineas de programa consecutivas */

const A = 1000 // el valor de la constante A está definido
A = 15;        // (ERROR! no se puede modificar el valor de la constante
```

Cada variable o constante debe tener un identificador que lo distingue de otras variables y constantes. Refiérase a los ejemplos anteriores, a y A son identificadores.

### Reglas para nombrar

En mikroC, los identificadores pueden ser tan largos como quiera. Sin embargo, hay varias restricciones:

- Los identificadores pueden incluir cualquiera de los caracteres alfabéticos A-Z (a-z), los dígitos 0-9 y el carácter subrayado '_'. El compilador es sensible a la diferencia entre minúsculas y mayúsculas. Los nombres de funciones y variables se escriben con frecuencia con minúsculas, mientras que los nombres de constantes se escriben con mayúsculas.
- Los identificadores no pueden empezar con un dígito.
- Los identificadores no pueden coincidir con las palabras clave del lenguaje mikroC, porque son las palabras reservadas del compilador.

El compilador mikroC reconoce 33 palabras clave:

| MIKROC - PALABRAS CLAVE | | | | |
|---|---|---|---|---|
| absolute | data | if | return | typedef |
| asm | default | inline | rx | typeid |
| at | delete | int | sfr | typename |
| auto | do | io | short | union |
| bit | double | long | signed | unsigned |
| bool | else | mutable | sizeof | using |
| break | enum | namespace | static | virtual |
| case | explicit | operator | struct | void |
| catch | extern | org | switch | volatile |
| char | false | pascal | template | while |
| class | float | private | this | |
| code | for | protected | throw | |
| const | friend | public | true | |
| continue | goto | register | try | |

Ejemplos de los identificadores válidos e inválidos:

```
temperatura_V1     // OK
Presión            // OK
no_corresponder    // OK
dat2string         // OK
SuMi               // OK
_texto             // OK
7temp              // NO -- no puede empezar con un número
%mas_alto          // NO -- no pueden contener caracteres especiales
if                 // NO -- no puede coincidir con una palabra reservada
j23.07.04          // NO -- no puede contener caracteres especiales (punto)
nombre de variable // NO -- no puede contener espacio en blanco
```

### Declaración de variables

Cada variable debe ser declarada antes de ser utilizada en el programa. Como las variables se almacenan en la memoria RAM, es necesario reservar el espacio para ellas (uno, dos o más bytes). Al escribir un programa, usted sabe qué tipo de datos quiere utilizar y qué tipo de datos espera como resultado de una operación, mientras que el compilador no lo sabe. No se olvide de que el programa maneja las variables con los nombres asignados. El compilador las reconoce como números en la memoria RAM sin conocer su tamaño y formato. Para mejorar la legibilidad de código, las variables se declaran con frecuencia al principio de las funciones:

```
<tipo> variable;
```

Es posible declarar más de una variable de una vez si tienen el mismo tipo.

```
<tipo> variable1, variable2, variable3;
```

Aparte del nombre y del tipo, a las variables se les asignan con frecuencia los valores iniciales justamente enseguida de su declaración. Esto no es un paso obligatorio, sino 'una cuestión de buenas costumbres'. Se parece a lo siguiente:

```
unsigned int peso; // Declarar una variable llamada peso
peso = 20;         // Asignar el valor 20 a la variable peso
```

Un método más rápido se le denomina declaración con inicialización (asignación de los valores iniciales):

```
unsigned int peso = 20; // peso está declarado y su valor es 20
```

Si hay varias variables con el mismo valor inicial asignado, el proceso se puede simplificar:

```
unsigned int peso1 = peso2 = peso3 = 20;
int valor_inicial = un_minimo_de_petroleo = 0;
```

- Tenga cuidado de no declarar la misma variable otra vez dentro de la misma función.
- Puede modificar el contenido de una variable al asignarle un nuevo valor tantas veces que quiera
- Al declarar una variable, siempre piense en los valores que la variable tendrá que contener durante la ejecución de programa. En el ejemplo anterior, peso1 no se puede representar con un número con punto decimal o un número con valor negativo.

### Declaración de constantes

Similar a las variables, las constantes deben ser declaradas antes de ser utilizadas en el programa. En mikroC, no es obligatorio especificar el tipo de constante al declararla. Por otra parte, las constantes deben ser inicializadas a la vez que se declaran. El compilador reconoce las constantes por su prefijo const utilizado en la declaración. Dos siguientes declaraciones son equivalentes:

```
const int MINIMUM = -100; // Declaracion constante MINIMUM
const MINIMUM = -100;     // Declaracion constante MINIMUM
```

Las constantes pueden ser de cualquier tipo, incluyendo cadenas:

```
const T_MAX = 3.260E1;                  // constante de punto flotante T_MAX
const I_CLASS = 'A';                    // constante caracter I_CLASS
const Mensaje = "Presione el botón IZQUIERDA"; // constante de cadena Mensaje
```

Las constantes de enumeración son un tipo especial de constantes enteras que hace un programa más comprensible al asignar los números ordinales a las constantes. Por defecto, el valor 0 se asigna automáticamente a la primera constante entre llaves, el valor 1 a la segunda, el valor 2 a la tercera etc.

```
enum surtidores {AGUA,GASOLEO,CLORO}; // AGUA = 0; GASOLEO = 1; CLORO = 2
```

Es posible introducir directamente el valor de una constante dentro de la lista de enumeraciones. El incremento se detiene al asignar un valor a un elemento de matriz, después se reinicia a partir del valor asignado. Vea el siguiente ejemplo:

```
enum surtidores {AGUA,GASOLEO=0,CLORO}; // AGUA = 0; GASOLEO = 0; CLORO = 1
```

Las constantes de enumeración se utilizan de la siguiente manera:

```
int Velocidad_de_ascensor
enum motor_de_ascensor {PARADA,INICIO,NORMAL=MÁXIMO};
Velocidad_de_ascensor = NORMAL; // Velocidad_de_ascensor = 2
```

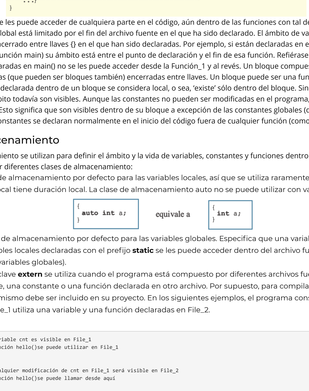
### Definir los nuevos tipos de datos

La palabra clave **typedef** le permite crear con facilidad los nuevos tipos de datos.

```
typedef unsigned int positivo; // positivo es un sinónimo para el tipo sin signo int
positivo a,b;                  // Variables a y b son de tipo positivo
a = 10;                        // Variable a equivale a 10
b = 5;                         // Variable b equivale a 5
```

### Ámbito de variables y constantes

Una variable o una constante es reconocida por el compilador en base de su identificador. Un identificador tiene significado si el compilador lo puede reconocer. El ámbito de una variable o una constante es el rango de programa en el que su identificador tiene significado. El ámbito es determinado por el lugar en el que se declara una variable o una constante. Intentar acceder a una variable o una constante fuera de su ámbito resulta en un error. Una variable o una constante es invisible fuera de su ámbito. Todas las variables y constantes que pensamos utilizar en un programa deben ser declaradas anteriormente en el código. Las variables y constantes pueden ser globales o locales. Una variable global se declara en el código fuente, fuera de todas las funciones, mientras que una variable local se declara dentro del cuerpo de la función o dentro de un bloque anidado en una función.

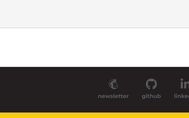



A las variables globales se les puede acceder de cualquiera parte en el código, aún dentro de las funciones con tal de que sean declaradas. El ámbito de una variable global está limitado por el fin del archivo fuente en el que ha sido declarado. El ámbito de variables locales está limitado por el bloque encerrado entre llaves {} en el que han sido declaradas. Por ejemplo, si están declaradas en el principio del cuerpo de función (igual que en la función main) su ámbito está entre el punto de declaración y el fin de esa función. Refiérase al ejemplo anterior. A las variables locales declaradas en main() no se les puede acceder desde la Función_1 y al revés. Un bloque compuesto es un grupo de declaraciones y sentencias (que pueden ser bloques también) encerradas entre llaves. Un bloque puede ser una función, una estructura de control etc. Una variable declarada dentro de un bloque se considera local, o sea, 'existe' sólo dentro del bloque. Sin embargo, las variables declaradas fuera del ámbito todavía son visibles. Aunque las constantes no pueden ser modificadas en el programa, siguen las mismas reglas que las variables. Esto significa que son visibles dentro de su bloque a excepción de las constantes globales (declaradas fuera de cualquier función). Las constantes se declaran normalmente en el inicio del código fuera de cualquier función (como variables globales).

### Clases de almacenamiento

Las clases de almacenamiento se utilizan para definir el ámbito y la vida de variables, constantes y funciones dentro de un programa. En mikroC se pueden utilizar diferentes clases de almacenamiento:

- **auto** es una clase de almacenamiento por defecto para las variables locales, así que se utiliza raramente. Se utiliza para definir que una variable local tiene duración local. La clase de almacenamiento auto no se puede utilizar con variables globales.




- **static** es una clase de almacenamiento por defecto para las variables globales. Especifica que una variable es visible dentro del archivo. A las variables locales declaradas con el prefijo **static** se les puede acceder dentro del archivo fuente (o sea se comportan como variables globales).
- **extern**: la palabra clave **extern** se utiliza cuando el programa está compuesto por diferentes archivos fuente. Esto le permite utilizar una variable, una constante o una función declarada en otro archivo. Por supuesto, para compilar y enlazar este archivo correctamente, el mismo debe ser incluido en su proyecto. En los siguientes ejemplos, el programa consiste en dos archivos: File_1 y File_2. El File_1 utiliza una variable y una función declaradas en File_2.

**File 1:**

```
extern int cnt;     // Variable cnt es visible en File_1
extern void hello(); // Función hello()se puede utilizar en File_1

void main(){
  PORTA = cnt++;    // Cualquier modificación de cnt en File_1 será visible en File_2
  hello();          // Función hello()se puede llamar desde aquí
}
```

**File 2:**

```
int cnt = 0;
void hello();

void hello() { // Modificaciones que afectan a la
.           // cnt en File_1 son visibles aquí
.
.
}
```