Aplicaciones de Sistemas Embebidos con Doble Núcleo

Sistemas Embebidos en Lenguaje C

UTN FRA
Departamento de Ingeniería Electrónica
Laboratorio de Sistemas Embebidos

30 de julio de 2024

Índice

1. Por qué C?
2. Tipos de datos
   - Enteros
     - Tipos
     - Estandarización
   - Flotantes
     - Tipos
     - Representación IEEE 754
   - Arrays
   - Punteros
     - Características
     - Operadores especiales
   - Estructuras
     - Declaración y typedef
     - Asignación y campos
     - Punteros
3. Condicionales
   - Generalidades
   - if
   - if ... else
   - if ... else if ... else
   - switch ... case
4. Bucles
   - while
   - do ... while
   - for
5. Funciones
   - Definición
   - Parámetros
   - Retorno
6. Referencias

- Lenguaje compilado.
- Determinista.
- Acceso directo a memoria, periféricos y hardware.
- Rápido y fácilmente optimizable.
- Mucho soporte en bibliotecas.
- Fácilmente portable a otros microcontroladores.
- Sintáxis sencilla.
- Soporte en la totalidad de microcontroladores.
- Permite un uso eficiente de memoria volátil y no volátil.

- Existen en variantes de 8, 16, 32 y 64 bits.
- Preceder la declaración del tipo de entero con la palabra unsigned hace que toda la capacidad de la variable sea para numeros positivos.
- Pueden representar 2^n números, siendo n la cantidad de bits.

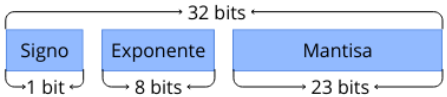
```
// Entero de 8 bits con signo
char foo;
// Entero de 16 bits con signo
short bar;
// Entero de 32 bits sin signo
unsigned int baz;
// Entero de 64 bits con signo
long int faz;
```

Podemos usar la biblioteca stdint.h que generalmente tenemos incluida para poder usar nombres más estándares para estos tipos de datos enteros.

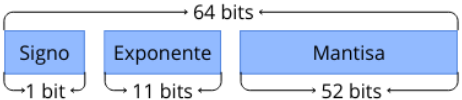
```
#include <stdint.h>

// Entero de 8 bits sin signo
uint8_t foo;
// Entero de 16 bits con signo
int16_t bar;
// Entero de 32 bits sin signo
uint32_t baz;
// Entero de 64 bits con signo
int64_t faz;
```

- Sirven para representar números con decimales.
- El tipo float es una variante de 32 bits con 7 decimales de precisión.
- El tipo double es una variante de 64 bits con 15 decimales de precisión.
- Siguen el estándar de punto flotante IEEE 754.
- El tipo float tiene un rango de valores de $3{,}4x10^{-38}$ a $3{,}4x10^{38}$.
- El tipo double tiene un rango de valores de $1{,}7x10^{-308}$ a $1{,}7x10^{308}$.

```
// Numero decimal de 32 bits
float foo = 3.1415926;
// Numero decimal de 64 bits
double bar = 2.718281828459045;
```



Float en IEEE 754



Double en IEEE 754

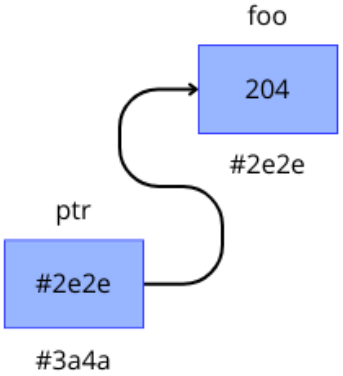
- Bit más significativo para el signo (0 positivo y 1 negativo).
- Los siguientes 8/11 bits se usan para almacenar el exponente con un offset ya que no se almacena directamente un exponente negativo.
- Los últimos 23/52 bits se usan para la mantisa, es decir, el número normalizado en notación científica.

- Es un conjunto de elementos del mismo tipo de dato.
- Un array de n elementos de b bytes ocupará en la memoria $n \times b$ bytes.
- La primera posición del array es la 0.
- Un array de n elementos tiene posiciones desde 0 hasta $n - 1$.

```
// Arrays de 4 elementos
int foo[4] = {9, 234, 7, -24};
int bar[] = {8, 54, -123, 33};
// Asignacion de elemento
foo[2] = 4096;
// Array inicializado en cero
int baz[4] = {0};
```

- Guardan la dirección de una variable.
- Con el operador **&** se puede ver la dirección de una variable, incluso un puntero.
- Con el operador ***** se puede ver el valor en la dirección que almacena el puntero.
- Se usan en pasajes por referencia.
- Permiten ver el valor de la dirección que guardan y modificarlo.
- Los arrays pueden usarse como punteros.

```c
// Creacion de variable entera
int foo = 204;
// Puntero a entero apuntando
// a la variable foo
int *ptr = &foo;
// Asignacion por referencia
*ptr = 400;
```



- Es un conjunto de elementos pero no necesariamente del mismo tipo de dato.
- Cada elemento que la compone se denomina campo y tiene un nombre y tipo de dato asociado.
- Usando la palabra typedef podemos definir a una estructura como un nuevo tipo de dato.

```
/**
 * @brief Definicion de tarea
*/
typedef struct {
    char *name;
    int priority;
    int stack_size;
    void (*callback)(void*);
} task_t;

// Creo variable del tipo task_t
task_t task;
```

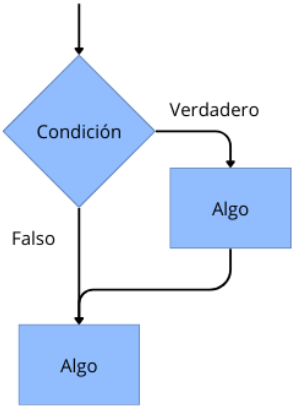
- Para acceder a un campo, se usa el operador **.** entre el nombre de la variable y el nombre del campo.
- Con los campos, se ordena mejor la información de un conjunto de datos.

```c
#include <string.h>

// Asigno al campo name
strcpy(task.name, "Task");
// Asigno al campo priority
task.priority = 1;
// Asigno al campo stack_size
task.stask_size = 1024;
```

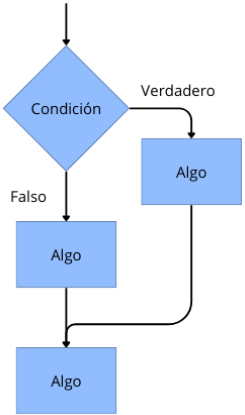
```c
/**
 * @brief Callbak para tarea
*/
void task_cbk(void *params) {
    // TODO
}

// Puntero a estructura
task_t *ptr = &task;
// Asigno callback
ptr->callback = task_cbk;
```

- Es posible crear punteros a estructuras que se definan.
- Cuando se maneja un puntero a estructura, para acceder a un campo se usa el operador $->$.

- La condición que se evalúa, se escribe entre paréntesis.
- Sólo importa si la condición o valor en el paréntesis termina siendo algo interpretable como verdadero o falso.
- Se pueden usar los operadores $==$, $>$, $>=$, $<$, $<=$ o $!=$ para comparar expresiones.
- Para concatenar múltiples condiciones, se usa el operador || (OR) o el operador && (AND).
- En el caso de los switch case, los casos se evalúan sólo por igualdad.

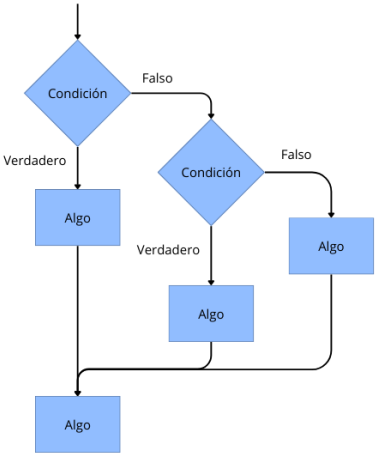


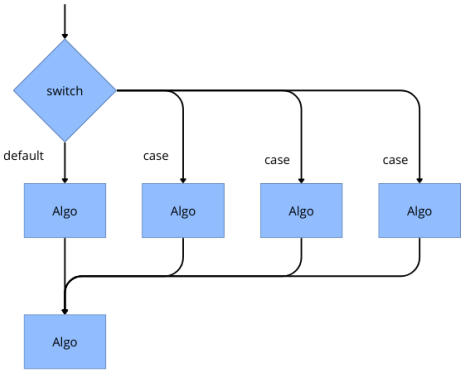

```c
if(foo > 10) {
    // Verdadero
    // Hago algo
}

// Falso
// Hago otra cosa
```




```
if(foo > 10) {
    // Verdadero
    // Hago algo
}
else {
    // Falso
    // Otra cosa
}
```

```
if(foo > 10) {
    // Verdadero
    // Hago algo
}
else if(foo < 0) {
    // Falso primero
    // Verdadero ahora
    // Hago otra cosa
}
else {
    // Ambos falsos
    // Otra cosa
}
```






```
switch(foo) {
    // Solo si foo==1
    case 1:
    // Algo
    break;
    // Solo si foo==2
    case 2:
    // Algo
    break;
    // Ninguno anterior
    default:
    // Otra cosa
    break;
}
```

- Repiten el código expresado en las llaves mientras que la condición sea verdadera.
- La condición se evalúa antes de entrar al bucle, si es falsa, no se ejecuta ninguna iteración.
- La condición luego se evalúa al comenzar cada nueva iteración.

```
// Evalua condicion
while(foo < 10) {
    // Verdadero
    // Hago cosas
}
```

```
do {
    // Hago cosas
    // Evaluo condicion
} while(foo < 10);
```

- La condición para iterar se escribe el final.
- A diferencia del while tradicional, la condición se evalúa al final de cada iteración.
- Son útiles donde por lo menos debe asegurarse una iteración.

- Se puede asignar un valor inicial a una variable (condición inicial) en el primer argumento del bucle.
- La condición para iterar se escribe en el segundo argumento.
- En cada iteración, se puede incrementar el valor de una variable. El incremento se describe en el tercer argumento.
- El incremento puede ser positivo o negativo.

```
// Condicion arranca en 0
// Evalua condicion
for(int i = 0; i < 10; i++) {
    // Hago cosas
    // Incremento variable
}
```

- Ayudan a encapsular y abstraer código.
- Son un bloque de código con nombre, valores de entraday valor de salida que se puede invocar cuando queramos.
- Pueden tener múltiples valores de entrada pero solo uno de salida.
- Si no devuelve o recibe valores, se define con la palabra void.
- Normalmente, se usa un prototipo, es decir, un descriptor de la función, antes de invocarla.

- Son variables que recibe la función.
- Pueden usarse para operar dentro de la función.
- Se guardan en el stack.
- No existe límite a cuantos parámetros puede tener una función.

```
// Prototipo de funcion
void foo(char bar, int baz);
```

```
/**
 * @brief Ejemplo de funcion
 * @param bar caracter
 * @param baz numero
*/
void foo(char bar, int baz) {
    // Imprimo por consola
    printf("Recibi-%c-y-%d\n",
        bar, baz
    );
}
// Invoco funcion
foo('F', 4);
```

```c
// Prototipo de funcion
float sq(float base);

/**
 * @brief Ejemplo de funcion
 * @param base numero
 * @return cuadrado del numero
*/
float sq(float base) {
    return base * base;
}

// Invoco funcion
float res = sq(2.4);
```

- El valor de retorno puede ser solo uno.
- La palabra return se usa para describir que valor o variable de la función se desea retornar.
- Al retornar, se termina la ejecución de la función.
- El retorno de una variable es la única forma de obtener algún valor que fue usado localmente en la función.
- Es posible guardar el retorno de una función en otra variable.

Algunos recursos útiles

- IEEE-754 Floating Point Converter
- Aritmética de punto fijo
- Pasaje por referencia
- Aritmética de punteros
- Estructuras
- Condicionales
- Bucles
- Argumentos y retorno
- Argumentos vs Parámetros