Introducción a C

Juan Nehuen Gonzalez Montoro
Pablo Recabarren

Cátedra de Electrónica Digital III

15 de Agosto de 2013

"C"

- Es un lenguaje estructurado
- Es un lenguaje no interpretado
- Es un lenguaje de nivel medio
- El componente básico de C es la "Función"

Compilación

Al programar en C, generalmente el proceso que se sigue es el siguiente:

Edición del código

En C tenemos dos tipos de archivos,

- .c Archivos compilables. Sólo puede haber uno por ejecutable o que contenga un main. Cada programa puede utilizar tantos como se requiera.
- .h Archivos no compilables. Suelen incluirse en los .c para usar definiciones y prototipos.

Compilación

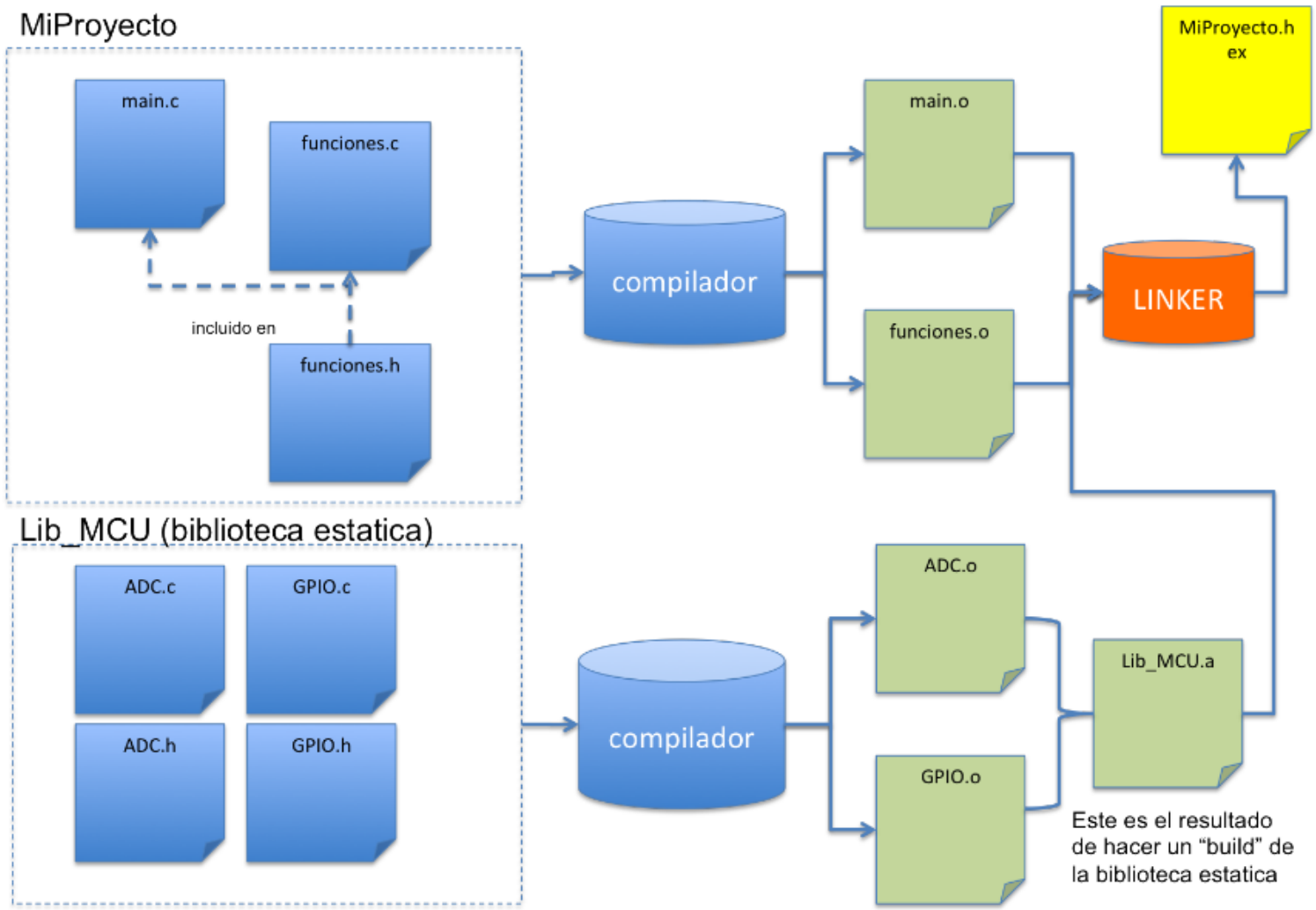
Al querer generar un programa ejecutable. Primero necesitaremos ejecutar el compilador. Este antes de empezar a compilar, ejecuta el pre-procesador.
Algunas de las tareas que este realiza son:

- Inclusión de archivos.
- Compilación condicional.
- Interpretación de macros y definiciones.
- Comprobación de errores de sintaxis.

Compilación

Ahora el compilador es quien se encarga de generar código objeto.
Incluir archivos .h (o prototipos de funciones) en el archivo .c le indican al compilador que las funciones existen, tienen un prototipo determinada y en algún momento serán linkeadas. Es decir que el código tendrá llamadas a las funciones pero no su implementación.

Cuando se tienen todos los archivos .o, se puede proceder al linkeo unión de los archivos para generar un archivo ejecutable. La salida generalmente es un archivo .axf o .elf que contienen el código ejecutable. Luego se puede pasar a un archivo .hex


MiProyecto
main.c
funciones.c
incluido en
funciones.h
Lib_MCU (biblioteca estatica)
ADC.c
GPIO.c
ADC.h
GPIO.h
compilador
compilador
main.o
funciones.o
ADC.o
GPIO.o
LINKER
MiProyecto.hex
Lib_MCU.a
Este es el resultado de hacer un "build" de la biblioteca estatica

Identificadores

- Nombre simbólico que se refiere a un dato, variable o función
- El nombre debe estar relacionado con el dato al que representa
- Los identificadores no pueden ser palabras reservadas
- Deben ser declarados por el usuario, indicando el nombre y el tipo de dato que van a representar

Identificadores

- Un identificador se forma con letras minúsculas, mayúsculas y dígitos.
- El carácter subrayado _ se considera como una letra más.
- Un identificador no puede contener espacios en blanco
- El primer carácter de un identificador debe ser siempre una letra o un _
- Se hace distinción entre letras mayúsculas y minúsculas.
- ANSI C permite definir identificadores de hasta 31 caracteres

Palabras reservadas

abstract	event	namespace	static
as	explicit	new	string
base	extern	null	struct
bool	false	object	switch
break	finally	operator	this
byte	fixed	out	throw
case	float	override	true
catch	for	params	try
char	foreach	private	typeof
checked	goto	protected	uint
class	if	public	ulong
const	implicit	readonly	uncheked
continue	in	ref	unsafe
decimal	int	return	ushort
default	interface	sbyte	using
delegate	internal	seales	virtual
do	is	short	volatile
double	lock	sizeof	void
else	long	stackalloc	while
enum			

Tipos de datos

Conceptualmente, desde el punto de vista de un programador, una **variable** *es una entidad cuyo valor puede cambiar* a lo largo de la ejecución de un programa.
En un nivel más lógico, una variable ocupa un espacio de memoria reservado en el ordenador para contener sus valores durante la ejecución de un programa.
Cada variable debe pertenecer a un **tipo** determinado.

char

- Ocupa 8 bits (1 byte)
- Es tratado como un entero para las operaciones matemáticas
- Rango de -128 a 127 en su versión con signo
- Rango de 0 a 255 en su versión sin signo

```
1 signed char cuenta, cuenta2, total;
2 unsigned char letras;
3 char caracter, inicial, respuesta;
4 signed char _letra;
```

int

- Ocupa 32 bits
- Puede modificarse usando short (16 bits), long(64 bits) o long long(128 bits)
- Puede declararse signed o unsigned

```
5  int cuenta, cuenta2, total; //signed
6  unsigned int suma;
7  signed int inicial, respuesta;
8  long sumaLarga;
9  unsigned long unsignedSumaLarga;
10 long long sumaMuyLarga;
```

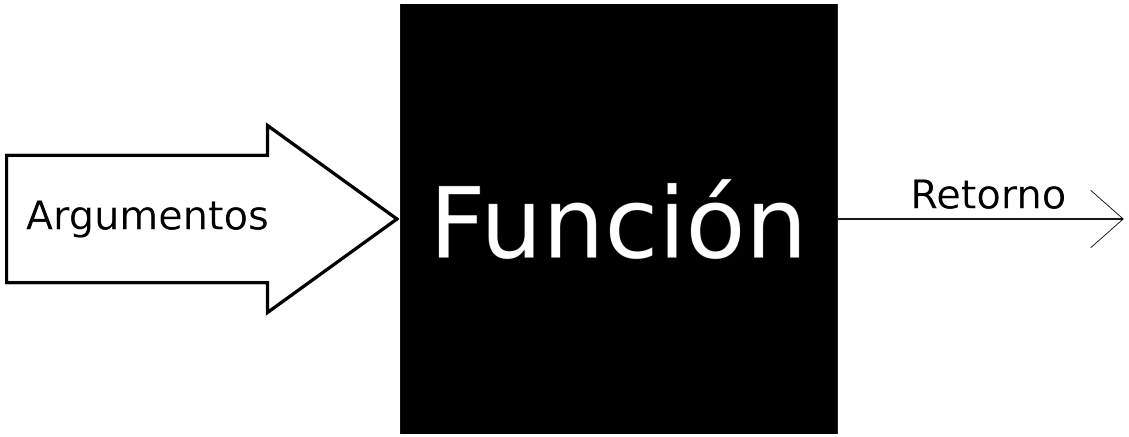
float

- Almacena numeros de punto flotante de 32 bits
- Rango de -990000000 y 99000000

```
12 float a = 1233554562123215 4;
13 float b = 3;
14
15 a = a + 1;
16 a = a - 1233554562123215 4;
17
18 b = b + 1;
19 b = b - 3;
```

Funciones

Una función es un conjunto de instrucciones que realizan una tarea específica. En general toman valores de entrada, llamados **parámetros** y proporcionan un valor de salida o valor de **retorno**; aunque en C++, tanto unos como el otro son opcionales, y pueden no existir.



Declaración de Funciones

En C++ es obligatorio usar prototipos. Un prototipo es una **declaración** de una función. Consiste en una presentación de la función, exactamente con la misma estructura que la definición, pero sin cuerpo y terminada con un ";". La declaracion se compone por:

- El tipo del valor de retorno, que puede ser void, si no necesitamos valor de retorno.
- El identificador de la función. Es costumbre, muy útil y muy recomendable, poner nombres que indiquen, lo más claramente posible, qué es lo que hace la función, y que permitan interpretar qué hace el programa con sólo leerlos.
- Una lista de declaraciones de parámetros entre paréntesis. Los parámetros de una función son los valores de entrada (y en ocasiones también de salida). Para la función se comportan exactamente igual que variables, y de hecho cada parámetro se declara igual que una variable.

```
23 int Mayor(int a, int b);
24 int Mayor(int, int);
25 void mostrar();
```

Definición de Funciones

Una definicin contiene además de la declaración las instrucciones con las que la función realizar su trabajo, es decir, su código. La sintaxis es idéntica a la del prototipo, salvo que se elimina el punto y coma final, y se añade el cuerpo de función que representa el código que será ejecutado cuando se llame a la función. El cuerpo de la función se encierra entre llaves "{}".

```
27 int Mayor(int a, int b)
28 {
29         if(a>b) return a;
30         else return b;
31 }
```

Estructura del programa

directivas del preprocesador: includes y defines
declaración de variables globales
prototipos de funciones
función main
definiciones de funciones

Una función muy especial es la función **main**, se trata de la función de entrada, y debe existir siempre, ya será la que tome el control cuando se ejecute el programa.

Operadores Aritméticos


```
33 // Operadores Arit. unitarios
34 + expresi n
35 - expresi n
36
37 variable ++      //(post-incremento)
38 ++ variable      //(pre-incremento)
39 variable --      //(post-decremento)
40 -- variable      //(pre-decremento)
41
42 //Operadores Arit. binarios
43 expresi n + expresi n
44 expresi n - expresi n
45 expresi n * expresi n
46 expresi n / expresi n
47 expresi n % expresi n
```

Operadores Aritméticos


```
50
51 c = a + ++b;
52 c = a + b++;
53
54 b = b+1;
55 c = a + b;
56
57 c = a + b;
58 b = b+1;
```

Operadores de asignación


```
59
60  expresin  = expresin
61  expresin *= expresin
62  expresin /= expresin
63  expresin %= expresin
64  expresin += expresin
65  expresin -= expresin
```

Operadores de comparación


```
68
69 expresi n1 == expresi n2
70 expresi n1 != expresi n2
71 expresi n1 > expresi n2
72 expresi n1 < expresi n2
73 expresi n1 <= expresi n2
74 expresi n1 >= expresi n2
```

Operadores lógicos


```
76
77 expresion1 && expresion2
78 expresion1 || expresion2
79 !<expresion>
```

Operadores orientados a bit


```
81
82 expreson1 & expresion2
83 expresion1 | expresion2
84 ~ expresi n
85
86 expresion << constante
87 expresion >> constante
```

Bifurcaciones - Sentencias if

Permite ejecutar una sentencia simple o compuesta según se cumpla o no una determinada condición.
if(expresión)sentencia;sentencia;...;sentencia;

```
89
90 if(a==0)  a=1;
91
92
93 if(a!=1)
94 {
95           b=a+1;
96           c--;
97 }
```

Bifurcaciones - Sentencias if - else

Permite realizar una bifurcación, ejecutando una parte del código u otra en función de la condición.
if(expresión)sentencia;sentencia;...;sentencia; else sentencia;sentencia;....;sentencia;

```
99
100 if(a==0)  a=1;else  b++;
101
102 if(a!=1){
103           b=a+1;
104           c--;
105 }
106 else{
107           c++
108 }
```

Bifurcaciones - Sentencia if - else múltiple

Permite realizar una ramificación múltiple ejecutando una sección de código según se cumpla una entre n condiciones.

```
if (expresion_1)
sentencia_1;
else if (expresion_2)
sentencia_2;
else if (expresion_3)
sentencia_3;
else if (...)
...
else sentencia_n;
```

Bifurcaciones - Sentencia if - else múltiple

Se evalúa expresión y se considera el resultado de dicha evaluacin. Si dicho resultado coincide con el valor constante expresion_cte_i, se ejecuta la sentencia_i si luego sigue una sentencia break el resto de los casos no se evalúan.

```
switch (expresion)
case expresion_cte_1:
sentencia_1;
break;
case expresion_cte_2:
sentencia_2;
break;
...
case expresion_cte_n:
sentencia_n;
break;
default:
```

Bucle - Sentencia while

Permite ejecutar una sentencia simple o compuesta mientras se cumpla una condición.
while (expresion_de_control)
sentencia;

```
110
111 while(1)  a++;
112
113 a=0;
114 while(a<100)
115 {
116           a++;
117           b+=a+c;
118 }
```

Ciclo for

se ejecuta inicialización, luego se evalúa la expresion_de_control y si es false se prosigue en la sentencia siguiente a la construccin for; si es true se ejecutan sentencia y actualización, y se vuelve a evaluar expresion_de_control.

```
120
121 for(i=0;i<10;i++)a+=i;
122
123 for(u=1000,r=0;i>0;i/=10)
124 {
125         r+=u;
126 }
```

Ciclo do - while

Es similar al bucle while con la diferencia de que la condición se evalúa luego de ejecutar las sentencias, las sentencias se ejecutan siempre al menos una vez.

```
128
129 do
130 {
131             a++;
132             r+=(a*3);
133 }while(a<3);
```

Sentencias de salto

los programas C se ejecutan secuencialmente, pero existen formas de romper este orden secuencial

El uso de **break** dentro de un bucle, una sentencia de selección o de un bloque, transfiere la ejecución del programa a la primera sentencia que haya a continuacin

```
134
135 y = 0;x = 0;
136 while(x < 1000)
137 {
138     if(y == 1000) break;
139     y++;
140 }
141 x = 1;
```

Sentencias de salto

El uso de **continue** dentro de un bucle ignora el resto del cdigo de la iteración actual, y comienza con la siguiente, es decir, se transfiere la ejecucin a la evaluacin de la condicin del bucle.

```
143
144 y = 0;
145 x = 0;
146 while(x < 1000)
147 {
148     x++;
149     if(y >= 100) continue;
150     y++;
151 }
```

Sentencias de salto

return es la sentencia de salida de una función, cuando se ejecuta, se devuelve el control a la rutina que llamó a la función.

```
153
154 int Paridad(int x)
155 {
156     if(x % 2) return 1;
157     return 0;
158 }
```

declaración de variables

Toda variable debe ser declarada e inicializada antes de poder ser utilizada correctamente. Si por error usamos una variable que no ha sido declarada, se producirá un error de compilación.

```
160
161 unsigned int a, b=0;
162
163 char nombre;
164
165 float r=0.0;
166
167 a=1;
```

```
169 int EnteroGlobal;
170 int Funcion1(int a);
171
172 int main() {
173     int EnteroLocal;
174     EnteroLocal = Funcion1(10);
175     EnteroGlobal = Funcion1(EnteroLocal);
176     return 0;
177 }
178
179 int Funcion1(int a)
180 {
181     char CaracterLocal;
182     if(EnteroGlobal != 0) return a/EnteroGlobal
            ;
183     return 0;
184 }
```

Enmascaramiento

Al declarar una variable global y una local con un mismo nombre. La variable local enmascara a la global, el acceso a la variable global está bloqueado. Para acceder a la global es necesario utilizar el operador ámbito (::).

```
185 int x; // Variable global
186
187 int main()
188 {
189     int x; // Variable local que enmascara a la
            global
190     x = 10; // Accedemos a la variable local
191     ::x = 100; // Mediante el operador de \'
            ambito accedemos a la global
192     return 0;
193 }
```

Arreglos

Los arrays permiten agrupar datos usando un único identificador. Todos los elementos de un array son del mismo tipo, y para acceder a cada elemento se usan índices.

```
195
196 int arr[10];
197 char Mensaje[] = "Error de lectura";
198 float real[3]={2.0, 3.5, 4.9};
199 int [2][2]={}
200 int nElementos;
201 ...
202
203 arr[2]=0;
204 nElementos = sizeof(arr)/sizeof(arr[0]);
```

Estructuras

Al contrario que los arrays, las estructuras nos permiten agrupar varios datos, que mantengan algn tipo de relacin, aunque sean de distinto tipo, permitiendo manipularlos todos juntos, usando un mismo identificador, o cada uno por separado.

```
206 struct Persona {
207     char Nombre[65];
208     char Direccion[65];
209     int AnyoNacimiento;
210 } Fulanito;
211
212 Fulanito.AnyoNacimiento = 1988;
```

Estructuras


```
215 struct stPareja {
216     int A, B;
217     int LeeA() { return A;}
218     int LeeB() { return B;}
219     void GuardaA(int n) { A = n;}
220     void GuardaB(int n) { B = n;}
221 } Par;
222
223 int main() {
224     int a,b;
225     Par.GuardaA(15);
226     Par.GuardaB(63);
227     b = Par.LeeA();
228     a = Par.LeeB();
229
230     while(1);
231
232 }
```