

# 75.40 Algoritmos y Programación I Curso 4

## Introducción al Lenguaje C

Dr. Mariano Méndez  
Juan Pablo Capurro  
Martin Dardis  
Matias Gimenez

Facultad De Ingeniería. Universidad de Buenos Aires

19 de agosto de 2019

### 1. Un Poco de Historia

“C es un lenguaje de programación de propósito general que ofrece como ventajas economía de expresión, control de flujo y estructuras de datos modernos y un rico conjunto de operadores. Además, C no es un lenguaje de muy alto nivel ni grande, y no está especializado en alguna área de aplicación en particular. Pero su ausencia de restricciones y su generalidad lo hacen más conveniente y efectivo para muchas tareas que otros lenguajes supuestamente más poderosos. Originalmente, C fue diseñado para el sistema operativo Unix y Dennis Ritchie lo implantó sobre el mismo en la DEC PDP-11. El sistema operativo, el compilador de C y esencialmente todos los programas de aplicación de Unix (incluyendo todo el software para preparar este libro) están escritos en C. También existen compiladores para la producción en otras máquinas, incluyendo la IBM System/370, la Honeywell 6000 y la Interdata 8/32. El lenguaje C no está ligado a ningún hardware o sistema en particular y es fácil escribir programas que corran sin cambios en cualquier máquina que maneje C.” [3] prefacio de la primera edición.

El lenguaje de programación C fue forjado en los laboratorios Bell (AT&T) por Dennis M. Ritchie en los años 70. Basado originalmente en el lenguaje de programación B, cuyo mayor defecto era la carencia de tipos de datos predefinidos, Ritchie transformó el lenguaje de programación B en el lenguaje de programación C manteniendo la sintaxis de B y además agregándole tipos de datos predefinidos y otros cambios.



Figura 1: Ken Thompson (sentado) y Dennis Ritchie en la PDP-11

El desarrollo de C comenzó en 1972 y está fuertemente ligado al sistema operativo Unix. Vio la luz del día en la versión 2 de Unix. En 1978 Brian Kernighan y Dennis Ritchie publicaron la primera edición del libro “El lenguaje de programación C” [3] que sirvió durante mucho tiempo como especificación del lenguaje. Hasta que en 1989 el American National Standards Institute ratificó la primera versión del estándar del lenguaje de programación C conocido como ANSI C, Estándar C o C89. La última versión del estándar conocido es C11, publicada el 8 de diciembre de 2011.

## 2. Instalación

A continuación se explicará cómo instalar los programas necesarios para poder implementar programas en el lenguaje de Programación C. Para ello se utilizará el compilador Open Source más conocido del lenguaje C llamado GCC, originalmente GCC significaba GNU C Compiler (compilador GNU de C), porque compilaba solamente este lenguaje. Posteriormente se extendió para compilar C++, Fortran, Ada y muchos otros lenguajes de programación. Actualmente su nombre significa GNU Compiler Collection [4].

Si se está utilizando una PC que tiene instalado un sistema operativo basado en Linux o Unix se debe ejecutar, desde la línea de comandos, la siguiente instrucción para saber si efectivamente GCC está instalado:

```
$gcc -v
```

Si el compilador se encuentra instalado en esa computadora debería aparecer el siguiente mensaje:

```
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/4.8/lto-wrapper
Target: x86_64-linux-gnu
Configured with: ../src/configure -v -
--with-pkgversion='Ubuntu 4.8.2-19ubuntu1'
--with-bugurl=file:///usr/share/doc/gcc-4.8/README.Bugs
--enable-languages=c,c++,java,go,d,fortran,objc,obj-c++
--program-suffix=-4.8 --enable-shared --enable-linker-build-id
--host=x86_64-linux-gnu --target=x86_64-linux-gnu
Thread model: posix
gcc version 4.8.2 (Ubuntu 4.8.2-19ubuntu1)
```

De no aparecer ningún mensaje significa gcc no está instalado en la computadora, por lo tanto se deberá proceder a la instalación manual. La forma más genérica de instalar gcc se encuentra en <http://gcc.gnu.org/install/>. Si contamos con una de las instalaciones de linux más comunes, como por ejemplo, Ubuntu podremos instalarlo desde la línea de comando de una terminal escribiendo:

```
$ sudo apt-get install gcc
```

lo que nos permitirá instalar la última versión disponible de gcc para nuestra versión del sistema operativo.

## 3. El Sistema de Compilación

Las partes del proceso de compilación son 4 fases, que utilizan 4 herramientas para llevar a cabo cada una de estas fases. Estas herramientas son:

- el preprocesador.
- el compilador.
- el ensamblador.
- el link-editor o linker.

Estas en conjunto son conocidas como un sistema de compilación.

### 3.1. La Compilación

- **La fase de procesamiento.** El preprocesador (cpp) modifica el código de fuente original de un programa escrito en C de acuerdo a las directivas que comienzan con un caracter(#). El resultado de este proceso es otro programa en C con la extensión .i
- **La fase de compilación.** El compilador (cc) traduce el programa .i a un archivo de texto .s que contiene un programa en lenguaje assembly.
- **La fase de ensamblaje.** A continuación el ensamblador (as) traduce el archivo .s en instrucciones de lenguaje de máquina completándolas en un formato conocido como programa objeto realocable. Este es almacenado en un archivo con extensión .o

- **La fase de link edición.** Generalmente los programas escritos en lenguaje C hacen uso de funciones que forman parte de la **biblioteca estándar de C** que es provista por cualquier compilador de ese lenguaje. Por ejemplo la función `printf()`, la misma se encuentra en un archivo objeto precompilado que tiene que ser mezclado con el programa que se esta compilando, para ello el linker realiza esta tarea teniendo como resultado un archivo objeto ejecutable.

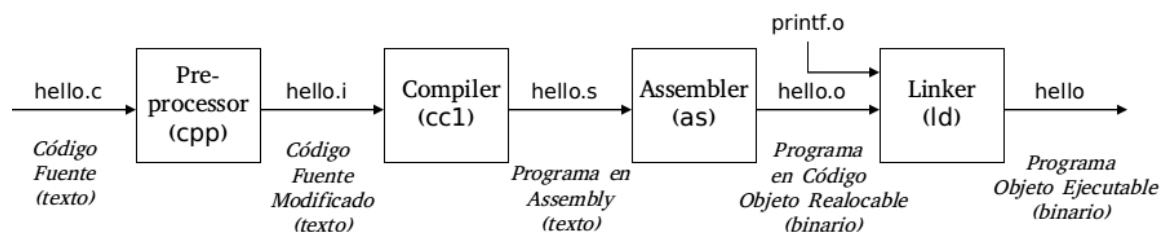


Figura 2

1. Programador edita código fuente
2. El compilador compila el source code en una secuencia de instrucciones de máquina y datos llamada.
3. El compilador genera esa secuencia y posteriormente se guarda en disco: programa ejecutable.

En este punto se tiene un programa capaz de ser ejecutado por una computadora, es función del sistema operativo hacer que este se ejecute.

## 4. El lenguaje C

### 4.1. Palabras Reservadas

Según la sección 3.1.1 del American National Standard for Information Systems — Programming Language C, X3.J11-1988 [1] las 32 palabras reservadas del lenguaje de programación C se listan a continuación:

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

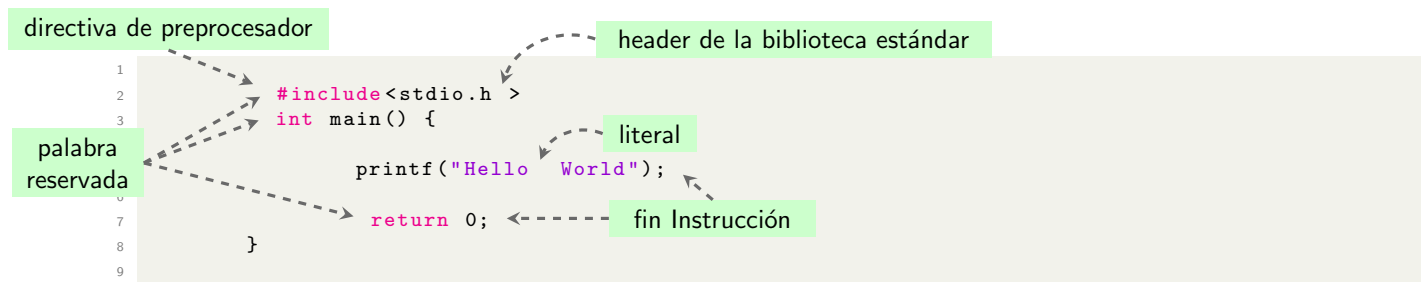
Según [1] los identificadores tal cual están listados anteriormente serán palabras reservadas y no podrán ser utilizados para ningún otro propósito.

### 4.2. Estructura de un Programa C vs. Pseudocódigo

En esta sección se muestra un primer ejemplo de programa escrito en el lenguaje de programación C.

<pre> 1 Algoritmo HolaMundo 2 const 3   //definicion de constantes 4 var 5   //definicion de variables 6 inicio 7   escribir("Hello World!"); 8 fin </pre>	<pre> 1 #include&lt;stdio.h&gt; 2 int main() { 3     printf("Hello World!"); 4     return 0; 5 } </pre>
--	---

Una pequeña explicación de qué es cada una de las cosas que se ven en el programa escrito en lenguaje C:



- `#include <stdio.h>` : Es la directiva de preprocesador necesaria para poder usar `printf()` . En pseudocódigo se omite la inclusión de bibliotecas por simplicidad, y por ese mismo motivo se usa `escribir()` en vez de las complejas funciones de entrada/salida de C.
- `main` : Es la función que se invocará cuando comience el programa, es decir, la ejecución del código comenzará allí. Los caracteres `{}` delimitan el código a ejecutar de forma análoga a los `inicio` y `fin` del pseudocódigo.
- `return 0;` : Ordena que el programa termine su ejecución y devuelva `0` al contexto en el que fue ejecutado. Esta funcionalidad no está contemplada en el pseudocódigo de la cátedra.

## 5. Tipos básicos de datos

El lenguaje C provee varios tipos básicos de datos, la mayoría de estos están formados por uno de los cuatro especificadores de tipos aritméticos básicos del lenguaje C (`char`, `int`, `float` y `double`).

### 5.1. Tipo char

Las variables carácter (tipo `char`) contienen un único carácter y se almacenan en un byte de memoria (8 bits). Como se vió el byte es una unidad de medida correspondiente a 8 bits, un bit (Binary Digit) es la mínima unidad de medida de la información.

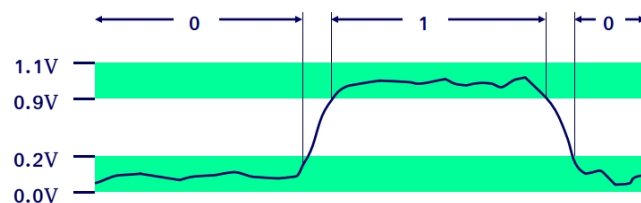


Figura 3: Un bit desde el punto de vista de la Física

La cantidad de combinaciones posibles que se pueden obtener con  $j$  bits está dada por la fórmula  $2^j$ , por ende con un byte se pueden obtener  $2^8$  combinaciones, es decir, 256 valores posibles.

0 1 1 0 1 1 1 1

Byte

$$0 * 2^7 + 1 * 2^6 + 1 * 2^5 + 0 * 2^4 + 1 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 = 101$$

en la tabla ascii está asignado al carácter "e"

binario	Dec	hex	caracter	binario	Dec	hex	caracter	binario	Dec	hex	caracter
0011 0000	48	30	0	0100 0100	68	44	D	0110 0001	97	61	a
0011 0001	49	31	1	0100 0101	69	45	E	0110 0010	98	62	b
0011 0010	50	32	2	0100 0110	70	46	F	0110 0011	99	63	c
0011 0011	51	33	3	0100 0111	71	47	G	0110 0100	100	64	d
0011 0100	52	34	4	0100 1000	72	48	H	0110 0101	101	65	e
0011 0101	53	35	5	0100 1001	73	49	I	0110 0110	102	66	f
0011 0110	54	36	6	0100 1010	74	4A	J	0110 0111	103	67	g
0011 0111	55	37	7	0100 1011	75	4B	K	0110 1000	104	68	h
0011 1000	56	38	8	0100 1100	76	4C	L	0110 1001	105	69	i
0011 1001	57	39	9	0100 1101	77	4D	M	0110 1010	106	6A	j
0100 0001	65	41	A	0100 1110	78	4E	N	0110 1011	107	6B	k
0100 0010	66	42	B	0100 1111	79	4F	O	0110 1100	108	6C	l
0100 0011	67	43	C	0101 0000	80	50	P	0110 1101	109	6D	m

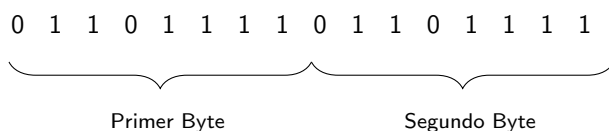
## 5.2. Tipo de datos enteros

En este caso veremos los tipos de datos que surgen del especificador de tipo aritmético básico int y también de la combinación de especificadores opcionales (signed, unsigned, short, long).

Tipo	Tamaño	Mínimo	Máximo
[signed]char	1 byte	-128	127
unsigned char	1 byte	0	255
short	2 bytes	-32,768	32,767
short int			
signed short			
signed short int			
unsigned short	2 bytes	0	65,535
unsigned short int			
[signed] int	4 bytes	-2,147,483,648	2,147,483,647
unsigned int	4 bytes	0	4,294,967,295
long	4 bytes	-2,147,483,648	2,147,483,647
long int			
signed long			
signed long int			
unsigned long	4 bytes	0	4,294,967,295
unsigned long int			
int32_t	4 bytes	-2,147,483,648	2,147,483,647
uint32_t	4 bytes	0	4,294,967,295
int64_t	4 bytes	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
uint64_t	8 bytes	0	18,446,744,073,709,551,615

Cuadro 1: Rangos de tipos de datos de C en una arquitectura de 32 bits

Para saber cómo se obtienen los valores máximos y mínimos de cada tipo de dato, hay que remitirse a la cantidad de combinaciones posibles según la longitud en bytes del mismo. Vale la pena destacar que los tipos de datos marcados como signed utilizan el bit más significativo para almacenar el signo, un 1 corresponde a negativo y un 0 corresponde a positivo. En el caso de un short int (entero corto) de 2 bytes:



Y para determinar el rango de valores se aplica una de las siguientes fórmulas:

Tipo	Tamaño	Mínimo	Máximo
[signed] char	1 byte	-128	127
unsigned char	1 byte	0	255
short	2 bytes	-32,768	32,767
short int			
signed short			
signed short int			
unsigned short	2 bytes	0	65,535
unsigned short int			
[signed] int	4 bytes	-2,147,483,648	2,147,483,647
unsigned int	4 bytes	0	4,294,967,295
long	8 bytes	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
long int			
signed long			
signed long int			
unsigned long	8 bytes	0	18,446,744,073,709,551,615
unsigned long int			
int32_t	4 bytes	-2,147,483,648	2,147,483,647
uint32_t	4 bytes	0	4,294,967,295
int64_t	8 bytes	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
uint64_t	8 bytes	0	18,446,744,073,709,551,615

Cuadro 2: Rangos de tipos de datos de C en una arquitectura de 64 bits

- Si el tipo de dato se define unsigned, el rango de valores va de  $0 \dots 2^k - 1$ , donde  $k$  es la longitud en bits.
- Si el tipo de dato se define signed, el rango de valores va de  $-2^{k-1} \dots 2^{k-1} - 1$ , donde  $k$  es la longitud en bits.

El lenguaje de programación C tiene una referencia a los valores límites máximos y mínimos de cada tipo de dato, estos valores pueden ser encontrados en `<limits.h>`. Más adelante se utilizará.

### 5.3. Tipo booleano

El tipo de dato booleano se define como uno que sólo admite dos valores posibles: **true (verdadero)** y **false (falso)**. El lenguaje de programación C no incluye *out of the box* un tipo de dato booleano, en cambio, aquellas expresiones que evalúen a **cero** se las considera *falsas*, y a todas las demás, verdaderas.

Es posible abstraerse de esta implementación poco intuitiva mediante la inclusión de `stdbool.h`, una biblioteca del estándar, que permite declarar variables de tipo `bool` y asignarles `true` o `false`.

### 5.4. Tipo de datos de coma flotante

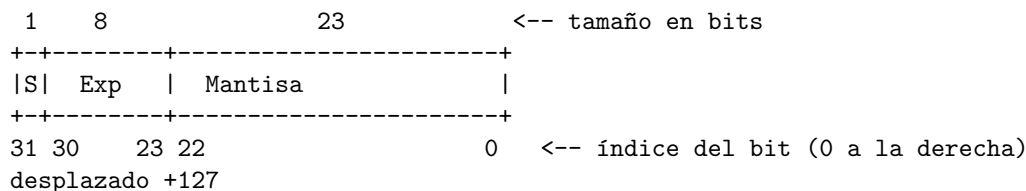
La representación de coma flotante (en inglés floating point) es una forma de notación científica usada en los microprocesadores con la cual se pueden representar números racionales extremadamente grandes y pequeños de una manera muy eficiente y compacta, y con la que se pueden realizar operaciones aritméticas. El estándar para la representación en coma flotante es el IEEE 754 [2]. La forma normal de expresar estos números en decimal es  $\pm \text{mantisa} * 10^{\text{exponente}}$ . En una computadora la forma de escribir estos números es  $\pm \text{mantisa} * 2^{\text{exponente}}$ . Es importante destacar que la comparación de flotantes puede resultar problemática, debido a errores de redondeo, variables que se espera que sean iguales pueden tener valores ligeramente distintos. *Por esto, se debe evitar la 'comparación desnuda' de flotantes mediante el operador `==`, con técnicas que serán explicadas más adelante.*

Tipo	Tamaño	Rango de valores	Precisión
float	4 byte	1.2E-38 to 3.4E+38	6 lugares decimales
double	8 byte	2.3E-308 to 1.7E+308	15 lugares decimales
long double	10 byte	3.4E-4932 to 1.1E+4932	19 lugares decimales

El estándar define formatos para la representación de números en coma flotante (incluyendo el cero) y valores desnormalizados, así como valores especiales como infinito y NaN (Not a Number). IEEE 754 especifica cuatro formatos para la representación de valores en coma flotante: precisión simple (32 bits), precisión doble (64 bits), precisión simple extendida (43 bits, no usada normalmente) y precisión doble extendida (79 bits, usualmente implementada con 80 bits).

## 5.5. Tipo float

Este tipo de dato corresponde al formato de precisión simple de 32 bits:



A continuacion se muestran algunos ejemplos:

0	00000000	000000000000000000000000	=	0
1	00000000	000000000000000000000000	=	-0

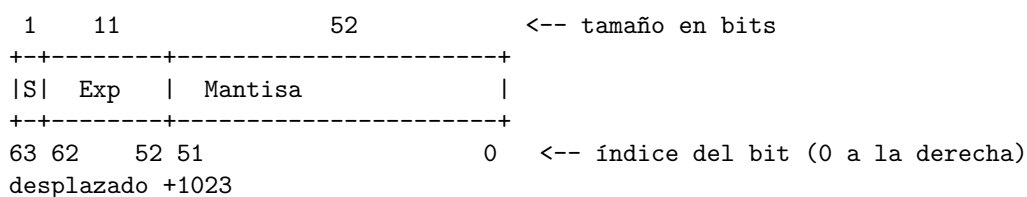
$$\begin{array}{llll} 0 & 11111111 & 000000000000000000000000 & = \infty \\ 1 & 11111111 & 000000000000000000000000 & = -\infty \end{array}$$

```
0  11111111  00000100000000000000000000000000  =  NaN
1  11111111  00100010001001010101010101010101  =  NaN
```

$$\begin{array}{llll} 0 & 10000000 & 00000000000000000000 & = +1 * 2^{128-127} * 1,0 = 2 \\ 0 & 10000001 & 1010000000000000000000 & = +1 * 2^{129-127} * 1,101 = 6,5 \\ 1 & 10000001 & 1010000000000000000000 & = -1 * 2^{129-127} * 1,101 = -6,5 \end{array}$$
$$\begin{array}{llll} 0 & 00000001 & 000000000000000000000000 & = +1 * 2^{1-127} * 1,0 = 2^{-126} \\ 0 & 00000000 & 10000000000000000000000000 & = +1 * 2^{126} * 0,1 = 2^{127} \\ 0 & 00000000 & 00000000000000000000000001 & = +1 * 2^{-126} * 2^{-23} = 2^{-149} \end{array}$$

## 5.6. Tipo double

Este tipo de dato corresponde al formato de precisión doble de 64 bits:



Para tener una idea de la magnitud que se representa se analiza un ejemplo, dado el número  $3FD55555555555_{16}$  en binario =  $0011111111001_2$ , tenemos entonces según la representación

$$\text{Signo} = 0$$
$$\text{Exponente} = 3FD16_{16} = 1021_{10}$$

Mantisa = 5 5555 5555 555516

$$\text{Número} = 2^{(\text{Exponente}-1023)} \times 1.\text{mantisa}$$
$$= 2^{-2} \times (1555555555555516 \times 2^{-52})$$
$$= 2^{-54} \times 1555555555555516$$
$$= 0.33333333333333314829616256247390992939472198486328125$$
$$approx = 1/3$$

## 6. Variables

Para definir una variable en C se debe establecer el tipo de dato al cual pertenece, para ello se utiliza cualquiera de los especificadores de tipos de las tablas de la sección anterior. Seguidamente se debe dotar de un nombre a la variable, a ese nombre se lo denomina **identificador**. Un identificador **se lo escribe teniendo en cuenta que es una secuencia de letras (letras minúsculas, letras mayúsculas, y el carácter “\_” es considerado una letra) y dígitos**. *Es importante recalcar que la primera letra de un identificador no puede ser un dígito numérico (0,1,2,3,4,5,6,7,8,9)*. La declaración se termina con un carácter “;”. Se hace distinción entre letras mayúsculas y minúsculas. Así, Casa es considerado como un identificador distinto de casa y de CASA.

```
1 int    numero;
2 long int contador;
3 char   letra;
4 float  raiz;
```

Ejemplos de identificadores válidos son los siguientes:

tiempo, distancia1, caso\_A, PI, velocidad\_de\_la\_luz

Por el contrario, los siguientes nombres no son válidos (¿Por qué?)

1\_valor, tiempo-total, dolare\$, %final

Cuando se desea realizar más de una declaración de variables del mismo tipo de dato son equivalentes:

<pre>1 int numero; 2 int contador; 3 int cantidadDePasos; 4 double raiz; 5 double importe;</pre>	<pre>1 int    numero, contador, cantidadDePasos; 2 double raiz, importe; 3 4</pre>
--	--

### 6.1. Operador sizeof()

Este operador devuelve como resultado el tamaño en bytes del tipo de dato o de la variable que se le pasase como parámetro. Por ejemplo:

```
1 int cuantos_bytes;
2
3 cuantos_bytes=sizeof(int);
```

El trozo de código fuente guarda en la variable `cuantos_bytes` la cantidad de bytes que ocupan las variables de tipo `int`.

## 7. Comentarios

Los caracteres `/*` se emplean para iniciar un comentario introducido entre el código del programa; el comentario termina con los caracteres `*/`. No se puede introducir un comentario dentro de otro. Todo texto introducido entre los símbolos de comienzo `/*` y final `*/` de comentario son siempre ignorados por el compilador. Por ejemplo:

```
1 variable_1 = variable_2; /* En esta línea se asigna a
2                          variable_1 el valor
3                          contenido en variable_2 */
```

Una fuente frecuente de errores no especialmente difíciles de detectar al programar en C, es el olvidarse de cerrar un comentario que se ha abierto previamente. El lenguaje ANSI C permite también otro tipo de comentarios, tomado de C++. Todo lo que va en cualquier línea del código detrás de la doble barra `//` y hasta el final de dicha línea, se considera como un comentario y es ignorado por el compilador. Para comentarios cortos, esta forma es más cómoda que la anterior, pues no hay que preocuparse de cerrarlos (el fin de línea actúa como cierre). Como contrapartida, si un comentario ocupa varias líneas hay que repetir la doble barra `//` en cada una de las líneas. Con este segundo procedimiento de introducir comentarios, el último ejemplo podría ponerse en la forma:

```
1 variable_1 = variable_2; // En esta línea se asigna a
2                          // variable_1 el valor
3                          // contenido en variable_2
```



## 8. Primer Programa en C

El clásico primer programa en lenguaje C es el siguiente:

```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("Hello World");
6     return 0;
7 }
```

Todos los programas ejecutables escritos en C tienen un punto de entrada, que es el lugar en el cual el programa comienza a ejecutarse, para el lenguaje C ese punto es `main()`. El programa irá ejecutando cada una de las líneas hasta llegar a su fin. Estas líneas están delimitadas por los caracteres `{` y `}`.

## 9. Estructuras de Control

En esta sección se estudiarán las estructuras de control que proporciona el lenguaje de programación C realizando una comparación con las estructuras de control vistas en pseudocódigo.

### 9.1. Bloque de Acciones

Los bloques de acciones son agrupaciones de acciones o declaraciones que sintácticamente son equivalentes a una acción sencilla, permitiendo asegurar que si el bloque se ejecuta todas sus instrucciones se ejecutarán respetando la secuencia de las mismas dentro del bloque. Cabe recordar que las acciones se delimitan con el carácter `;`. En el lenguaje de programación C los caracteres `{}` delimitan el inicio `{` y el fin `}` de un bloque.

```
1 inicio
2     x =1;
3     y =2;
4     escribir(x+1);
5 fin
```

..

```
1 {
2     x =1;
3     y =2;
4     printf("%i",x+1);
5 }
```

### 9.2. Estructuras de control selectivas

Estas estructuras de control permiten alterar el flujo normal del programa según se cumpla o no una determinada condición. Esta condición está contenida en la expresión que está entre paréntesis. El resultado de evaluar esta expresión puede ser `true` (verdadero) o `false` (falso). En C el valor `true` está representado por un valor distinto de cero y `false` corresponde al 0.

#### 9.2.1. if simple

<pre>1 si (expresion) entonces accion;</pre>	<pre>1 if (expresion) accion;</pre>
--	-------------------------------------

En su defecto utilizando un bloque de acciones:

<pre>1 si (expresion) entonces 2 inicio 3     accion1; 4     accion2; 5     accion3; 6     ... 7     accionj; 8 fin</pre>	<pre>1 if (expresion){ 2     accion1; 3     accion2; 4     accion3; 5     ... 6     accionj; 7 }</pre>
---	--

#### 9.2.2. if - else

<pre>1 si (expresion) entonces 2     accion1; 3 sino 4     accion2;</pre>	<pre>1 if (expresion) 2     accion1; 3 else 4     accion2;</pre>
---	--

En su defecto utilizando un bloque de acciones:

```

1 si (expresion) entonces
2     inicio
3         accion1 ;
4         accion2;
5         accion3;
6         ...
7         accionj;
8     fin
9 sino
10    inicio
11        accionj+1;
12        accionj+2;
13        accionj+3;
14        ...
15        accionj+n;
16    fin

```

```

1 if (expresion){
2     accion1;
3     accion2;
4     accion3;
5     ...
6     accionj;
7 }else{
8     accionj+1;
9     accionj+2;
10    accionj+3;
11    ...
12    accionj+n;
13 }

```

### 9.2.3. if - else if

La construcción de esta forma de la estructura de control si (if) se la denomina si múltiple (if múltiple) o si anidados;

```

1 si (expresion) entonces
2     accion1
3 sino si (expresion2) entonces
4     accion2
5 sino si (expresion3) entonces
6     accion3
7 sino si (expresion4) entonces
8     accion4
9 sino
10    accion5;

```

```

1 if (expresion)
2     accion1;
3 else if (expresion2)
4     accion2;
5 else if (expresion3)
6     accion3;
7 else if (expresion4)
8     accion4;
9 else accion5;

```

**Análisis:** Se evalúa expresion1. Si el resultado es `true`, se ejecuta accion1. Si el resultado es `false`, se evalúa expresion2. Si el resultado es `true`, se ejecuta accion2, mientras que si es `false` se evalúa expresion3 y así sucesivamente. Si ninguna de las expresiones evalúa a `true`, se ejecuta accion5 que es la opción por defecto. Todas las acciones pueden ser simples o compuestas. Por ejemplo:

```

1 /* El siguiente programa mostrara por pantalla si el valor de una
2 variable entera es cero, negativo o positivo*/
3
4 #include<stdio.h>
5
6 int main(){
7     int mi_numero=10;                                // Se crea una variable con nombre mi_numero y
                                                         // se le asigna el valor 10
8
9     if (mi_numero == 0)
10        printf("mi_numero vale 0");
11     else if (mi_numero > 0)
12        printf("mi_numero es positivo");
13     else if (mi_numero < 0)
14        printf("mi_numer es negativo");
15
16     return 0;
17 }

```

### 9.2.4. switch - case

Esta estructura de control surge debido a situaciones en las cuales se deben realizar un bloque de acciones, o al menos una acción diferente, para cada valor/es posible/es de una variable. C proporciona una estructura de control que permite codificar en una solución a este problema. Lógicamente se podría seguir utilizando un `if - else if`, pero el código rápidamente perdería claridad al repetir múltiples veces siempre la misma pregunta: cuál es el valor de la misma variable. Con un `switch` se puede agrupar a todos los `if` s en una estructura clara que muestra sencillamente toda la información relevante de lo que esta sucediendo.

Ejemplo de uso:

```

1 switch (variable) {
2     case valor1:
3         accion1;    //Bloque de acciones que se
4                     //ejecuta si variable==valor1
5     break;
6
7     case valor2:
8         accion2;    //Bloque de acciones que se
9                     //ejecuta si variable==valor2
10    break;
11
12    ...
13    ...
14
15    case valorN:
16        accionN;    //Bloque de acciones que se
17                    //ejecuta si variable==valorN
18    break;
19
20    default:
21        accionDefault; //Bloque de acciones que se ejecuta
22                        //si variable guarda un valor
23                        //distinto a todos los anteriores
24    break;
25 }

```

Nótese que si bien los valores fueron enumerados, no se requiere que estos sean sucesivos. Es decir, podría ser :

```

1 switch (variable) {
2     case 5:
3         accionOBloqueDeAccionesQueSeEjecutaSiVariableEsIgualA5
4     break;
5
6     case 3:
7         accionOBloqueDeAccionesQueSeEjecutaSiVariableEsIgualA3
8     break;
9     case 7:
10        accionOBloqueDeAccionesQueSeEjecutaSiVariableEsIgualA7
11    break;
12 }

```

**Nota:** Mas allá de las posibilidades del lenguaje, para facilitar la lectura los case deberían seguir un orden lógico, y las condiciones ser constantes nombradas o funciones booleanas.

**Nota:** El caso default es puramente opcional, y no se requiere para el funcionamiento del case.

### 9.3. Estructuras de control iterativas

Las estructuras de control iterativas o repetitivas, repiten un bloque de acciones mientras se cumple una condición, o bien una cantidad determinada de veces. Este tipo de estructura de control suele denominarse en inglés *loop* y en español *ciclo* o *bucle*.

#### 9.3.1. While

Esta estructura de control repite una acción o bloque de acciones mientras la expresión de control evalúe a `true`. Esta evaluación se realiza antes de la ejecución del bloque. Funciona de la siguiente manera:

1. Se evalúa la expresión booleana de control de la iteración.
2. Si la evaluación arroja `true`, se ejecuta el bloque de acciones y se vuelve al paso anterior.
3. Si no, se continúa con la ejecución de la acción siguiente al bloque de acciones.

Ejemplo del uso de la estructura de control while de una sola acción:

```

1 Mientras (expresion) Hacer accion;
1 while (expresion) accion;

```

Ejemplo del uso de la estructura de control while repitiendo un bloque de acciones:

```

1 Mientras (expresion)  hacer
2 Inicio
3     accion1;
4     accion2;
5     ...
6     accionj;
7 Fin

```

```

1 while (expresion){
2     accion1;
3     accion2;
4     ...
5     accionj;
6 }

```

A continuación se muestra un sencillo ejemplo escrito en lenguaje C:

```

1 /* El siguiente programa muestra por pantalla el valor de una variable entera
2 sumada a si misma mientras que el valor de esta sea menor a 100 */
3
4 #include<stdio.h>
5
6 int main(){
7     int mi_numero=1;
8
9     while (mi_numero < 100) {
10         mi_numero= mi_numero + mi_numero;
11     }
12     printf("valor final de mi_numero es %i\n",mi_numero);
13
14     return 0;
15 }

```

### 9.3.2. do - while

Esta estructura de control repite una acción o bloque de acciones mientras la expresión de control resulte `true`. Pero en este caso, evalúa dicha expresión una vez terminada la ejecución, es decir que se garantiza que su acción o bloque de acciones se ejecute por lo menos una vez. Ejemplo del uso de la estructura de control `do - while` de una sola acción:

<pre> 1 Repetir 2     accion 3 Hasta (!expresion); </pre>	<pre> 1 do 2     accion; 3 while (expresion); </pre>
---	--

Ejemplo del uso de la estructura de control `do - while` repitiendo un bloque de acciones:

<pre> 1 Repetir 2     accion1; 3     accion2; 4     ... 5     accionj; 6 Hasta(expresion); </pre>	<pre> 1 do{ 2     accion1; 3     accion2; 4     ... 5     accionj; 6 }while (!expresion); </pre>
---	--

### 9.3.3. for

Esta estructura de control se utiliza cuando se conoce la cantidad de iteraciones previamente y esta cantidad es un número finito de repeticiones (iteración definida). Ejemplo del uso de la estructura de control `for` de una sola acción:

<pre> 1 Para variable &lt;- expresion1 hasta expresion2 2     Hacer accion; </pre>	<pre> 1 for (expresion1 ; expresion2 ; 2     expresion3 ) accion; </pre>
--	--

Ejemplo del uso de la estructura de control `for` repitiendo un bloque de acciones:

<pre> 1 Para Contador &lt;- expresion1 hasta expresion2 2     Hacer 3     Inicio 4         accion1; 5         accion2; 6         ... 7         accionj; 8     Fin </pre>	<pre> 1 for (expresion1 ; expresion2 ; 2     expresion3 ) { 3     accion1; 4     accion2; 5     ... 6     accionj; 7 } </pre>
--	---

Dado que la estructura de control `for` puede parecer compleja a continuación se propone un pequeño ejemplo;

```

1 Para contador ← 1 hasta 8 Hacer
2 Inicio
3     suma ← suma + contador ;
4     total ← total + suma ;
5 Fin

```

```

1 for (int contador = 1 ; contador <=8
    ; contador=contador+1 ) {
2     suma = suma + contador ;
3     total= total + suma ;
4 }

```

**Nota:** Cada una de las tres expresiones que intervienen en el `for` tienen un propósito bien específico:

- La `expresion1` se encarga de definir cuál es la variable de control del ciclo y el valor inicial de la misma. Puede definirse una variable en la misma expresión (compilando bajo el estandar C99, con `-std=c99`).
- La `expresion2` define cuál será el valor de corte del ciclo. Allí se define el valor final. Se debe tener cuidado dado que esta es una expresión booleana.
- La `expresion3` define cómo se incrementa la variable de control.

Otro ejemplo más de una iteración escrita con un `for`:

```

1 /*Suma una variable a si misma una cantidad de veces determinada*/
2 #include<stdio.h>
3
4 int main(){
5     int cantidad_de_sumas=10;
6     int mi_numero=1;
7
8     for(int i=0; i<=cantidad_de_sumas;i= i+1){
9         mi_numero=mi_numero + mi_numero;
10    }
11
12    return 0;
13 }

```

Es interesante destacar que la estructura de control `for` es totalmente equivalente a la siguiente construcción utilizando la estructura de control `while`:

```

1 for (expresion1 ; expresion2 ; expresion3 )
2     accion;

```

```

1 expresion1;
2 while (expresion2){
3     accion;
4     expresion3;
5 }

```

## 10. Operadores

Un operador es un caracter o grupo de caracteres que actúa sobre una, dos o más variables para realizar una determinada operación con un determinado resultado.

### 10.1. Operadores aritméticos

Operador	Descripción	Ejemplo
*	Multiplicación	(a*b)
/	División	(a/b)
+	Suma	(a+b)
-	Resta	(a-b)
%	Módulo	(a %b)

### 10.2. Operadores relacionales

Operador	Descripción	Ejemplo
<	Menor que	(a<b)
<=	Menor que o igual	(a <= b)
>	Mayor que	(a > b)
>=	Mayor o igual que	(a >= b)
==	Igual	(a == b)
!=	No igual	(a != b)

### 10.3. Operadores lógicos

Operador	Descripción	Ejemplo
expresion1    expresion2	or	(x>2)    (y<4)
expresion1 && expresion2	and	(x>2) && (y<4)

### 10.4. Operadores incrementales

Operador	Descripción	Ejemplo	equivalencia
++	incremento	i++	i=i+1
—	decremento	i—	i=i-1

### 10.5. Operadores de bits

Operador	Descripción	Ejemplo
&	AND bit a bit	C = A&B;
	OR inclusiva bit a bit	C = A B;
^	OR exclusiva bit a bit	C = A^B;
<<	Desplazar bits a izquierda	C = A<<B;
>>	Desplazar bits a derecha	C = A>>B;
—	Complemento a uno	C = -B;

### 10.6. Operadores de asignación

Operador	Descripción	Ejemplo
=	asignación simple	a = b;
*=	z * = 10;	z = z * 10;
/=	z / = 5;	z = z / 5;
%=	z % = 2;	z = z % 2;
+=	z + = 4;	z = z + 4;
-=	z - = 5;	z = z - 5;
<<=	z << = 3;	z = z << 3;
>>=	z >> = 4;	z = z >> 3;
&=	z & = j;	z = z & j;
^=	z ^ = j;	z = z ^ j;
—=	z — = j;	z = z — j;

## 11. Indentación

Según Martin Fowler *“cualquier tonto puede escribir un programa que una computadora pueda entender, los buenos programadores escriben programas que los seres humanos pueden entender”*. Esta frase que puede parecer trivial envuelve en sí misma uno de los grandes problemas que incluso hoy en día es un tema de estudio. La claridad y legibilidad de un programa. Normalmente, al conjunto que conforman todas las líneas de un programa se lo denomina Código Fuente o Source Code, inglés. Una computadora no necesita que el source code esté escrito de una manera especial, de hecho un programa cuyo código fuente esté escrito de la siguiente forma es totalmente comprensible por una computadora. Por ejemplo:

```
1 #include <stdio.h>
2 int main(void){ int i=0; printf("Hello, World!");
3 for (i=0; i<3; i++){ printf("\n"); } return 0;}
```

```
1 #include <string.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 main(1
5     ,a,n,d)char**a;{
6     for(d=atoi(a[1])/10*80-
7         atoi(a[2])/5-596;n=@"NKA\
8 CLCCGZAAQBEAADAFAISADJABBA^\
9 SNLGAQABDAXIMBACTBATAHDBAN\
10 ZcEMMCCCCAAhEIJFAEAAAABafHJE\
11 TBdFLDAANEfDNBPdHdBcBBBEA_AL\
12 H E L L O,      W O R L D! "
13     [1++-3];)for(;n-->64;)
14     putchar(!d+++33^
15             1&1);}
```

Otra posibilidad sería escribir el mismo programa de la siguiente forma, la cual hace mas legible y entendible el programa:

```
1 #include <stdio.h>
2 int main(void)
3 {
4     int i=0;
5     printf("Hello, World!");
6     for (i=0; i<1; i++)
7     {
8         printf("\n");
9     }
10    return 0;
11 }
```

De todas formas sigue tomando tiempo poder descifrar cuál es el propósito del programa. Una forma normal entre los programadores para poder hacer más legible el código fuente es aplicando la técnica llamada indentación. La indentación es un anglicismo (de la palabra inglesa indentation) de uso común en informática; no es un término reconocido por la Real Academia Española (consultado en la vigesimosegunda edición). La Real Academia recomienda utilizar "sangrado". Este término significa mover un bloque de texto hacia la derecha insertando espacios o tabulaciones, para así separarlo del margen izquierdo y distinguirlo mejor del texto adyacente; en el ámbito de la imprenta, este concepto siempre se ha denominado sangrado o sangría (fuente: wikipedia). El mismo fragmento de código fuente anterior al cual se ha aplicado la técnica de indentación o sangrado queda:

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     int i=0;
6
7     printf("Hello, World!");
8     for (i=0; i<1; i++)
9     {
10         printf("\n");
11     }
12
13     return 0;
14 }
```

A lo largo de los años distintos estándares de sangrado han sido definidos para C. A continuación se muestran algunos:

```
1 if (x == y) {
2     x++;
3     foo();
4 } else {
5     x--;
6     bar();
7 }
```

(a) K&amp;R

```
1 if (x == y)
2 {
3     x++;
4     foo();
5 }
6 else
7 {
8     x--;
9     bar();
10 }
```

(b) Allman

```
1 if (x == y)
2 {
3     x++;
4     foo();
5 }
6 else
7 {
8     x--;
9     bar();
10 }
```

(c) Whitesmith

```
1 if (x == y)
2 {
3     x++;
4     foo();
5 }
6 else
7 {
8     x--;
9     bar();
10 }
```

(d) GNU

```
1 if (x == y)
2 {
3     x++;
4     foo();
5 }
6 else
7 {
8     x--;
9     bar();
10 }
```

(e) Horstman

```
1 if (x == y) {
2     x++;
3     foo();
4 }
5 else {
6     x--;
7     bar();
8 }
```

(f) Banner

## 12. Entrada y Salida de Datos

Normalmente en pseudocódigo se utilizan dos acciones; una para la lectura de valores desde algún dispositivo de entrada ( `LEER()` ) y otra para la escritura de valores en algún dispositivo de salida ( `ESCRIBIR()` ). A continuación

veremos los equivalentes en C. Para poder utilizar estas funciones se debe hacer referencia en el programa a la biblioteca estándar `stdio.h`. Esto se realiza mediante la directiva `#include <stdio.h>`.

### 12.1. Salida de Datos: `printf()`

Esta función se utiliza para realizar operaciones de salida. La misma recibe una cadena que determina el formato y una lista de variables o expresiones:

```
1 printf (const char* formato, ..);
```

La cadena de formato provee una descripción de la salida. Dicha cadena posee dos tipos de caracteres, los caracteres comunes, que serán impresos tal cual en el dispositivo de salida, y los caracteres de especificación de formato con marcadores determinados por caracteres de escape `%`. Estos especificadores de formato se asignan según la localización relativa y el tipo de salida que la función debe producir. La lista de variables o expresiones proporcionan los valores a escribir, teniendo en cuenta su orden relativo:

```
1 printf("Nombre %s, numero uno %d, real %5.2f.\n", "Juan", 12345, 3.14);
```

la salida obtenida será:

Nombre Juan, numero uno 12345, real 3.14.

A continuación se describen los marcadores de formato:

Formateador	Salida
%d ó %i	entero en base 10 con signo (int)
%u	entero en base 10 sin signo (int)
%o	entero en base 8 sin signo (int)
%x	entero en base 16, letras en minúscula (int)
%X	entero en base 16, letras en mayúscula (int)
%f	Coma flotante decimal de precisión simple (float)
%lf	Coma flotante decimal de precisión doble (double)
%e	La notación científica (mantisa / exponente)
%E	La notación científica (mantisa / exponente)
%c	caracter (char)
%s	cadena de caracteres (string)

### 12.2. Entrada de Datos: `scanf()`

De la misma forma que en pseudocódigo es posible asignarle un valor a una variable desde un dispositivo de entrada (teclado, por ejemplo) en C también lo es. Para ello se utiliza la función `scanf()`. Al igual que `printf()` la misma recibe una cadena de control y una lista de variables o expresiones:

```
1 scanf (const char* formato, ... );
```

La cadena control o formato provee una descripción del formato de entrada de los datos. A este formato serán convertidos los datos ingresados en el dispositivo de entrada. Estos caracteres de especificación de formato se obtienen con marcadores determinados por caracteres de escape `%`. Dichos especificadores de formato se asignan según su localización relativa. La lista de variables o expresiones proporcionan los valores a escribir, teniendo en cuenta su orden relativo:

```
1 scanf("%f", &gradosCentigrados);
```

Lo que obtendremos de esta acción es que se esperará que desde el dispositivo de entrada estándar se ingrese un valor, se lo convertirá utilizando la cadena de formato especificada.



Formateador	Salida
%d ó %i	entero en base 10 con signo (int)
%u	entero en base 10 sin signo (int)
%o	entero en base 8 sin signo (int)
%x	entero en base 16, letras en minúscula (int)
%X	entero en base 16, letras en mayúscula (int)
%f	Coma flotante decimal de precisión simple (float)
%f	Coma flotante decimal de precisión doble (double)
%e	La notación científica (mantisa / exponente)
%E	La notación científica (mantisa / exponente)
%c	caracter (char)
%s	cadena de caracteres (string)

### 12.3. Entrada de Datos: `getchar()`

Esta función lee un caracter del stdin, la entrada estándar del programa, que generalmente es el teclado.

```

1 #include <stdio.h>
2
3 int main () {
4     char un_caracter;
5
6     printf("Ingresar un caracter: ");
7     un_caracter = getchar();
8
9     printf("el caracter ingresado es : ");
10    putchar(un_caracter);
11
12    return(0);
13 }
```

**Nota:** ver bien la descripción en el man de linux.

### 12.4. Salida de Datos: `putchar()`

Esta función escribe un caracter en el stdout, la salida estándar del programa, que normalmente es el monitor.

```

1 #include <stdio.h>
2
3 int main () {
4     char un_caracter;
5
6     for(un_caracter = 'A' ; un_caracter <= 'Z' ; un_caracter++) {
7         putchar(un_caracter);
8     }
9
10    return(0);
11 }
```

## 13. Compilando el Primer Programa en C

Recordando el primer programa en C mostrado:

```

1 #include<stdio.h>
2 int main() {
3     printf("Hello World!");
4     return 0;
5 }
```

Para poder hacer que éste funcione en nuestra computadora se lo guardará como `primerprograma.c`. Para que dicho programa pueda ser ejecutado, en primer lugar debe ser compilado, para lo que se utilizará el compilador de C. Desde el directorio en el que fue guardado el archivo se ejecutará por línea de comandos:

```
gcc primerprograma.c
```

Esto debería generar un archivo llamado `primerprograma`. Para verificar que se haya creado, se inspeccionan los

contenidos del directorio mediante `ls`. Esto debería mostrar por pantalla que existen tanto `primerprograma.c` como `a.out` (el ejecutable).

para que el programa ejecutable tenga un nombre en especial se debe compilar de la siguiente forma:

```
gcc primerprograma.c -o primerprograma
```

Para ejecutar el programa recién creado, lo invocamos con `./primerprograma`.



```
temp — fish /Users/martindardis/temp — fish — 84x24
martindardis@MacBook-Pro-de-Martin ~/temp> gcc primerprograma.c -o primerprograma
martindardis@MacBook-Pro-de-Martin ~/temp> ./primerprograma
Hello World!
martindardis@MacBook-Pro-de-Martin ~/temp>
```

Figura 5

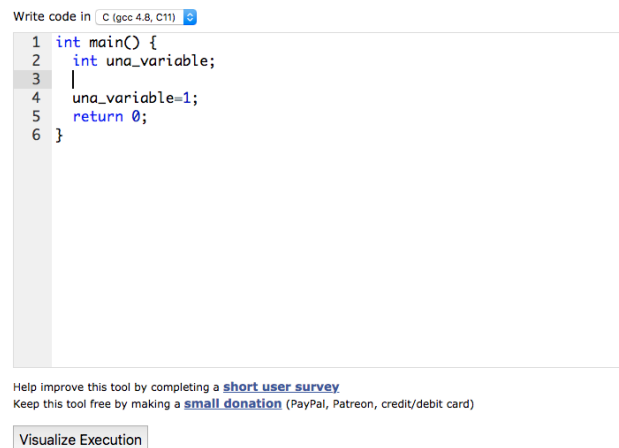
## 14. Herramientas de Consulta y para Practicar

Es verdad que no existen muchas herramientas que hagan comprender que está sucediendo en la computadora mientras se esta ejecutando un programa. Básicamente la mejor herramienta para ver esto es el debugger, que es demasiado complejo para utilizarlo en estas instancias. Por otro lado existe una herramienta llamada C Tutor, que puede ser localizada en <http://www.pythontutor.com/c.html#mode=edit>. Esta herramienta fue creada por Philip Guo (@pgbovine), para ayudar a romper la peor barrera que tiene un alumno cuando inicia a programar, que es saber que está pasando dentro de la computadora.

Por ejemplo:

```
1 int main() {
2     int una_variable;
3
4     una_variable=1;
5     return 0;
6 }
```

para utilizar la herramienta se ingresa a la pagina y se tipea el programa escrito en C:



Write code in C (gcc 4.8, C11)

```
1 int main() {
2     int una_variable;
3
4     una_variable=1;
5     return 0;
6 }
```

Help improve this tool by completing a [short user survey](#)  
Keep this tool free by making a [small donation](#) (PayPal, Patreon, credit/debit card)

Visualize Execution

Figura 6

A continuación se presiona **Visualize Execution**:

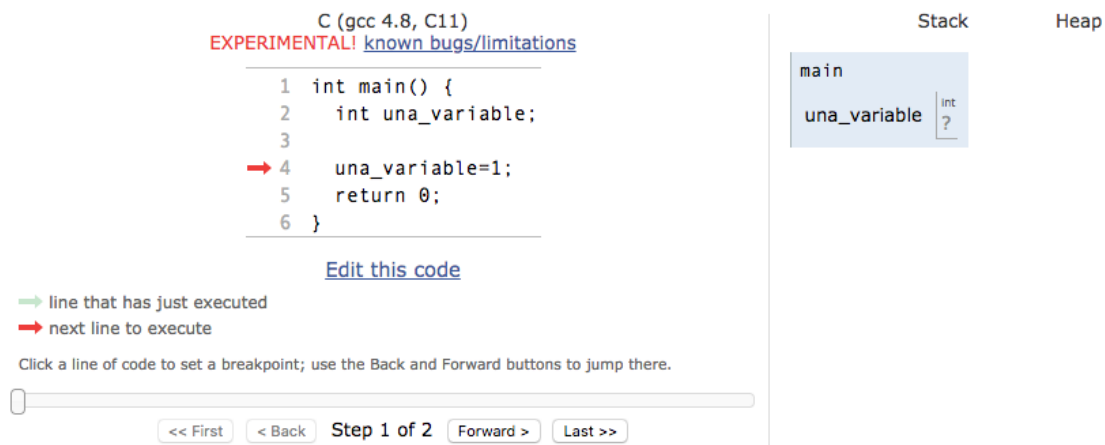


Figura 7: Primer Paso de Ejecución del Programa

En este primer paso se puede ver como se crea parte de la memoria de `main()` con su correspondiente variable, cuyo valor contenido es **basura (?)**!. La flecha roja indica cual es la línea próxima del programa a ser ejecutada, en este caso la línea 4.

A continuación se presiona **Forward>**, que indica que la línea debe ser ejecutada:



Figura 8: Segundo Paso de Ejecución del Programa

Puede verse que tras su ejecución el valor de la variable `una_variable`, pasa de tener basura a contener el valor 1, y la flecha roja **apunta a la próxima instrucción** del programa que debe ser ejecutada. En este caso la línea 5. Esta operación que se realiza con las variables, la de asignarle un valor inicial se denomina **inicialización de una variable**.

A continuación se continua presionando **Forward>** :

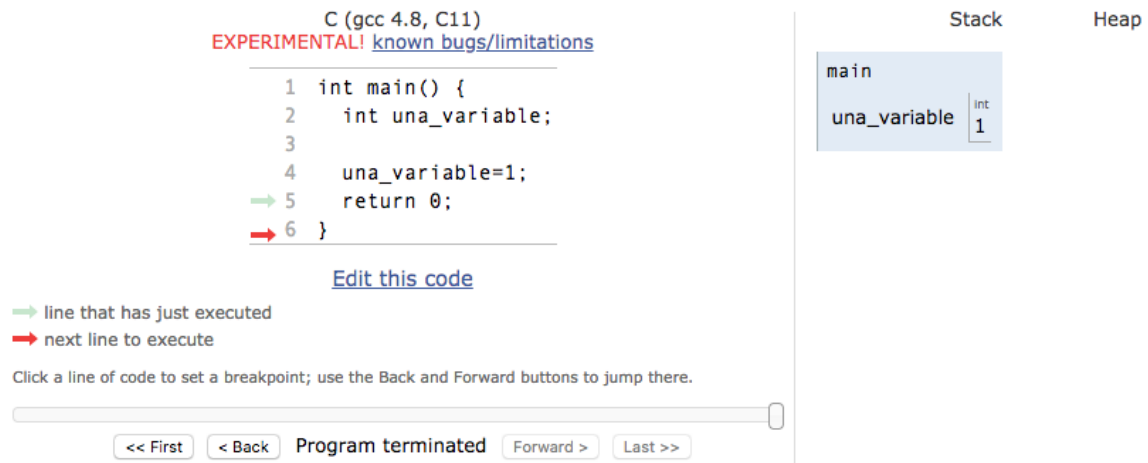


Figura 9: Tercer Paso de Ejecución del Programa

En este caso se ejecuta la instrucción `return 0`, que devuelve el valor 0 al sistema operativo. Y finalmente la última instrucción a ejecutarse es la salida del bloque de la función `main()`, cuyo resultado no es visible.

## 15. ¿Cómo Romperlo?

### 15.1. OverFlow

El término overflow se refiere en programación a cuando por algún motivo se quiere almacenar un número cuyo valor es más grande que el rango de valores posibles que puede almacenar una variable de un determinado tipo de dato.

```

1 int main() {
2     int una_variable;
3
4     una_variable=2147483647;
5     una_variable ++;
6     printf("El valor de la variable es : %d", una_variable)
7     return 0;
8 }

```

¿Qué salida tiene este fragmento de código?

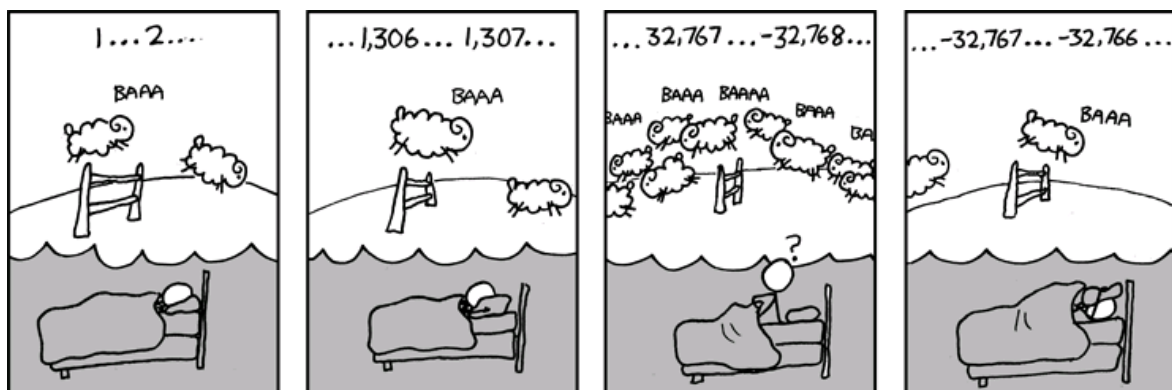


Figura 10: Overflow

## 16. Algunos ejemplos

En esta sección se muestran algunos ejemplos de programas que contiene entrada y salida de datos. Además se presentan algunos ejemplos básicos en los que se utilizan también las estructuras de control.

### 16.1. Entrada y salida de datos con formato

#### 16.1.1. Ejemplo: Lectura de un valor numérico por teclado

En este ejemplo se define una variable de tipo entera a la cual se le asigna un valor entero ingresado por teclado. Posteriormente se muestra el contenido de la variable por la salida estándar.

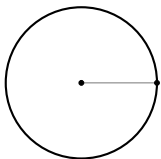
```

1 #include <stdio.h>
2
3 int main(){
4     int mi_variable;          /* declaracion de la variable */
5
6     /* recordar que antes de un scanf() siempre va un printf()*/
7
8     printf("ingrese un número entero:");    // escritura
9     scanf("%d",&mi_variable);              // lectura
10    printf("%d",mi_variable);
11    return 0;
12 }
```

#### 16.1.2. Ejemplo: Área y circunferencia

**Problema:** Diseñar un algoritmo que calcule la longitud de la circunferencia y el área de un círculo de radio dado.

##### 1. Análisis del problema:



$$\text{Longitud} = 2 * \pi * \text{radio}$$

$$\text{Area} = \pi * \text{radio}^2$$

**Datos de entrada:** Radio del círculo.

**Datos de salida:** Longitud y area del circulo.

*Ejemplo:*

$$\text{Radio} = 3$$

$$\text{Longitud} = 2 * \pi * \text{radio} = 2 * 3,1416 * 3 = 18,8496$$

$$\text{Area} = \pi * \text{radio}^2 = 3,1416 * 3^2 = 28,2744$$

##### 2. Diseño del algoritmo:

```

1 Algoritmo circulo;
2 Constantes
3     pi=3.1416;
4 Variables
5     radio:real;
6     longitud:real;
7     area:real;
8 inicio
9     longitud←0;
10    area←0;
11    escribir("ingrese el valor del radio del círculo");
12    leer(radio);
13    longitud←2*pi*radio;
14    area←pi * radio * radio;
15    escribir("La longitud del circulo es:",longitud);
16    escribir("El area del circulo es:",area);
17 fin.
```

### 3. Implementación en C:

```
1 #include <stdio.h>
2
3 int main(){
4     float radio;
5     float longitud;
6     float area;
7     float PI = 3.1416;
8
9
10    printf("ingrese el valor del radio del círculo");
11    scanf("%f",&radio);
12    longitud=2*PI*radio;
13    area=PI* radio * radio;
14    printf("La longitud del círculo es:%f",longitud);
15    printf("El área del círculo es:%f",area);
16    return 0;
17 }
```

## 16.2. Ejemplos varios

**Problema:** Se ingresa un texto por teclado, caracter a caracter, y se desea saber cuantas letras posee el texto.

1. **Análisis del problema:** Muchas veces realizar un buen análisis puede ser determinante para la resolución del problema. El enunciado de este problema suele llevar a las personas a creer que el texto debe ser almacenado completamente en una variable. Teniendo en cuenta los tipos de datos vistos hasta el momento, sería imposible de solucionar, pues no se ha visto aún, la forma de almacenar un texto entero en una variable. Aquí es donde una correcta interpretación del problema nos lleva a una solución simple y elegante.

- a) ¿Cómo se ingresa el texto? Caracter a caracter, por ende es necesario guardar en un caracter cada una de los símbolos del texto.
- b) ¿En algún lado del enunciado dice que el texto debe ser almacenado completo? No

### 2. Diseño del algoritmo:

```

1 Algoritmo letrasDeUnTexto;
2
3 Constante FIN='@';
4
5 Variables
6     letra:caracter;
7     cantidad:entero;
8
9 inicio
10    cantidad =0;
11    escribir("Ingrese el texto caracter a caracter y presione @ para finalizar");
12    leer(letra);
13    mientras (letra != FIN) hacer
14        inicio
15            si (letra<='A' && letra>='Z') || (letra<='a' && letra>='z') cantidad<-cantidad+1;
16            leer(letra);
17        fin
18    escribir("la cantidad de letras ingresadas es:", cantidad);
19 fin

```

### 3. Implementación en C:

```

1 #include <stdio.h>
2
3 int main() {
4     const char FIN = '@';
5
6     char letra='.';
7     int cantidad =0;
8
9     printf("Ingrese un texto letra por letra y termine con arroba\n");
10
11     while (letra!=FIN){
12         if( (letra<='A' && letra>='Z') || (letra<='a' && letra>='z') )
13             cantidad++;
14         letra=getchar();
15     }
16     printf("hay %d letras ingresadas \n", cantidad);
17     return 0;
18 }

```

**Problema:** Se ingresa un texto por teclado, caracter a caracter, y se desea saber cuantas letras posee el texto.

### 1. Análisis del problema:

- a) En ningún lugar menciona que el texto es guardado, solo se cuentan los caracteres.  
item Con saber cuantas letras del código ascii se ingresaron el problema estaría resuelto

### 2. Diseño del algoritmo:

```

1  Algoritmo letrasDeUnTexto;
2
3  Constante FIN='@';
4
5  Variables
6  letra:caracter;
7  cantidad:entero;
8
9  inicio
10 cantidad =0;
11 escribir("Ingrese el textto caracter a caracter y presione @ para finalizar");
12 leer(letra);
13 mientras (letra != FIN) hacer
14 inicio
15 si (letra<='A' && letra>='Z') || (letra<='a' && letra>='z') cantidad<-cantidad+1;
16 leer(letra);
17 fin
18 escribir("la cantidad de letras ingresadas es:", cantidad);
19 fin
20

```

### 3. Implementación en C:

```

1 #include <stdio.h>
2
3 int main() {
4     const char FIN = '@';
5
6     char letra='.';
7     int cantidad =0;
8
9     printf("Ingrese un texto letra por letra y termine con arroba\n");
10
11     while (letra!=FIN){
12         if( (letra<='A' && letra>='Z') || (letra<='a' && letra>='z') )
13             cantidad++;
14         letra=getchar();
15     }
16     printf("hay %d letras ingresadas \n", cantidad);
17     return 0;
18 }

```

## 17. Ejercicios Resueltos

### 17.1. Reparando la Nave

Durante una batalla espacial, la nave de Darth Méndez se averió. Desafortunadamente, su seguro intergaláctico no cubre siniestros ocurridos en batallas, por lo que deberá pagar el arreglo con sus ahorros. Se le pide realizar un algoritmo que usando las variables costo\_de\_reparacion (correspondiente a lo que cuesta arreglar los daños de la nave) y ahorros (correspondiente a los ahorros de Darth Mendez) imprima por pantalla si puede pagar el arreglo (mensaje : Tranki que podemos arreglarla) o no (mensaje : Siamo fuori, no podemos volver). Los mensajes deben ser exactamente esos.

#### 17.1.1. Posible Solución

```

1 #include <stdio.h>
2
3 int main(){
4     int costo_de_reparacion, ahorros;
5     scanf("%d", &costo_de_reparacion);
6     scanf("%d", &ahorros);

```



```

7
8     if(ahorros >= costo_de_reparacion){
9         printf("Tranki que podemos arreglarla\n");
10    }
11    else{
12        printf("Siamo fuori, no podemos volver\n");
13    }
14
15    return 0;
16 }

```

## 17.2. El Remis Milenario

A falta de dinero, a Han Solo se le está dificultando mantener el Halcón Milenario en condiciones óptimas. Debido a esto, decide usar dicha nave como taxi para llevar personas desde la cantina de Mos Eisley a distintos planetas. Para empezar, decide que solo va a transportar gente a planetas específicos y con tarifa fijas. Estos son: - Felucia (se identifica con 'F') con una tarifa de \$50. - Coruscant (se identifica con 'C') con una tarifa de \$80. - Bespin (se identifica con 'B') con una tarifa de \$120. Han necesita que crees un algoritmo que reciba el planeta al que quiere ir su pasajero (utilizar la variable destino) e imprima por pantalla "Su tarifa va a ser de N pesos" dependiendo del planeta al que quiera ir. En caso de que el planeta al que quiere ir no sea ninguno de los tres se deberá imprimir por pantalla "No puedo llevarte a ese planeta porque no está en la lista" Los mensajes deben ser exactamente esos.

### 17.2.1. Posible Solución

```

1 #include <stdio.h>
2
3 int main(){
4     char destino;
5     scanf("%c", &destino);
6
7     switch(destino){
8         case 'F':
9             printf("Su tarifa va a ser de 50 pesos");
10            break;
11            case 'C':
12                printf("Su tarifa va a ser de 80 pesos");
13                break;
14                case 'B':
15                    printf("Su tarifa va a ser de 120 pesos");
16                    break;
17                    default:
18                        printf("No puedo llevarte a ese planeta porque no esta en la lista");
19                        break;
20            }
21            return 0;
22 }

```

## 17.3. Modificando los Sables

A lo largo de múltiples batallas y entrenamientos, muchos stormtroopers sufrieron de ataques de epilepsia. Recientemente, un oficial de alto rango realizó un informe en el que demostró que los sables láser que NO son de color rojo eran los responsables de dichos ataques. Con este dato en mente, Darth Vader informó que todos los sables láser pertenecientes al imperio iban a ser modificados insertándoles un chip especial para cambiar su color a ROJO. Según lo establecido, los stormtroopers entraran de a 3 al cuarto de control de color de sables. Se pide realizar un algoritmo que, usando las variables color\_sable\_1, color\_sable\_2 y color\_sable\_3 que representan el color del sable (R: Rojo, V: verde), imprima por pantalla el texto: Se han cambiado N sables del color VERDE a ROJO Por ejemplo, si el sable 1 y 2 son verdes, se debe imprimir: Se han cambiado 2 sables del color VERDE a ROJO En caso de no haber ningún sable a modificar, se debe imprimir: Todos los sables son de color ROJO Los mensajes deben ser exactamente esos.

### 17.3.1. Posible Solución

```

1 #include <stdio.h>
2
3 const char VERDE = 'V';
4 const char ROJO = 'R';

```

```
5
6 int main(){
7     char color_sable_1, color_sable_2, color_sable_3;
8     int sables_cambiados = 0;
9     scanf(" %c", &color_sable_1);
10    scanf(" %c", &color_sable_2);
11    scanf(" %c", &color_sable_3);
12
13
14
15    if(color_sable_1 == VERDE)
16        sables_cambiados++;
17    if(color_sable_2 == VERDE)
18        sables_cambiados++;
19    if(color_sable_3 == VERDE)
20        sables_cambiados++;
21
22    if(sables_cambiados != 0)
23        printf("Se han cambiado %d sables del color VERDE a ROJO" , sables_cambiados);
24    else
25        printf("Todos los sables son de color ROJO");
26
27    return 0;
28 }
```

## 17.4. Estableciendo Comunicación

Debido a que el imperio cuenta con tecnología de punta para interceptar las comunicaciones, los rebeldes decidieron usar unos viejos dispositivos de comunicación llamados Walkie-Talkie. Los rebeldes acordaron una frecuencia por la cual comunicarse (variable frecuencia\_deseada). Realizar un algoritmo que dado un Walkie-Talkie encendido con una frecuencia inicial (variable frecuencia\_inicial) lo sintonice con la frecuencia deseada e imprima por pantalla "Después de N cambios de frecuencia, se logró sintonizar la correcta" Aclaraciones: La frecuencia inicial del Walkie-Talkie siempre es menor que la FRECUENCIA DESEADA. La frecuencia del Walkie-Talkie debe aumentarse de a intervalos de 0.1

### 17.4.1. Posible Solución

```
1 #include <stdio.h>
2
3 int main(){
4     float frecuencia_deseada, frecuencia_inicial, frecuencia_actual;
5     scanf("%f", &frecuencia_deseada);
6     scanf("%f", &frecuencia_inicial);
7
8     int cambios_de_frecuencia=0;
9     frecuencia_actual=frecuencia_inicial;
10
11    while(frecuencia_actual<frecuencia_deseada){
12        frecuencia_actual=(frecuencia_actual+0.1);
13        cambios_de_frecuencia++;
14    }
15
16    printf("Despues de %d cambios de frecuencia, se logró sintonizar la correcta",
17    cambios_de_frecuencia);
18    return 0;
19 }
```

## 18. Tabla ASCII

Decimal	Hexa	Caracter	Decimal	Hexa	Caracter	Decimal	Hexa	Caracter
000d	00h	(nul)	048d	30h	0	096d	60h	'
001d	01h	(soh)	049d	31h	1	097d	61h	a
002d	02h	(stx)	050d	32h	2	098d	62h	b
003d	03h	(etx)	051d	33h	3	099d	63h	c
004d	04h	(eot)	052d	34h	4	100d	64h	d
005d	05h	(enq)	053d	35h	5	101d	65h	e
006d	06h	(ack)	054d	36h	6	102d	66h	f
007d	07h	(bel)	055d	37h	7	103d	67h	g
008d	08h	(bs)	056d	38h	8	104d	68h	h
009d	09h	(tab)	057d	39h	9	105d	69h	i
010d	0Ah	(lf)	058d	3Ah	:	106d	6Ah	j
011d	0Bh	(vt)	059d	3Bh	;	107d	6Bh	k
012d	0Ch	(np)	060d	3Ch	i	108d	6Ch	l
013d	0Dh	(cr)	061d	3Dh	=	109d	6Dh	m
014d	0Eh	(so)	062d	3Eh	¿	110d	6Eh	n
015d	0Fh	(si)	063d	3Fh	?	111d	6Fh	o
016d	10h	(dle)	064d	40h	@	112d	70h	p
017d	11h	(dc1)	065d	41h	A	113d	71h	q
018d	12h	(dc2)	066d	42h	B	114d	72h	r
019d	13h	(dc3)	067d	43h	C	115d	73h	s
020d	14h	(dc4)	068d	44h	D	116d	74h	t
021d	15h	(nak)	069d	45h	E	117d	75h	u
022d	16h	(syn)	070d	46h	F	118d	76h	v
023d	17h	(etb)	071d	47h	G	119d	77h	w
024d	18h	(can)	072d	48h	H	120d	78h	x
025d	19h	(em)	073d	49h	I	121d	79h	y
026d	1Ah	(eof)	074d	4Ah	J	122d	7Ah	z
027d	1Bh	(esc)	075d	4Bh	K	123d	7Bh	-
028d	1Ch	(fs)	076d	4Ch	L	124d	7Ch	—
029d	1Dh	(gs)	077d	4Dh	M	125d	7Dh	"
030d	1Eh	(rs)	078d	4Eh	N	126d	7Eh	~
031d	1Fh	(us)	079d	4Fh	O	127d	7Fh	△
032d	20h	(space)	080d	50h	P	128d	80h	€
033d	21h	!	081d	51h	Q	129d	81h	,
034d	22h	"	082d	52h	R	130d	82h	,
035d	23h	#	083d	53h	S	131d	83h	f
036d	24h	\$	084d	54h	T	132d	84h	"
037d	25h	%	085d	55h	U	133d	85h	...
038d	26h	&	086d	56h	V	134d	86h	†
039d	27h	'	087d	57h	W	135d	87h	‡
040d	28h	(	088d	58h	X	136d	88h	^
041d	29h	)	089d	59h	Y	137d	89h	‰
042d	2Ah	*	090d	5Ah	Z	138d	8Ah	Š
043d	2Bh	+	091d	5Bh	[	139d	8Bh	<
044d	2Ch	,	092d	5Ch	"	140d	8Ch	Œ
045d	2Dh	-	093d	5Dh	]	141d	8Dh	
046d	2Eh	.	094d	5Eh	^	142d	8Eh	Ž
047d	2Fh	/	095d	5Fh	·	143d	8Fh	

Decimal	Hexa	Caracter	Decimal	Hexa	Caracter	Decimal	Hexa	Caracter
144d	90h		181d	B5h	μ	218d	DAh	Ú
145d	91h	‘	182d	B6h	¶	219d	DBh	Û
146d	92h	,	183d	B7h	.	220d	DCh	Ü
147d	93h	“	184d	B8h	/	221d	DDh	Ý
148d	94h	”	185d	B9h	ı	222d	DEh	þ
149d	95h	•	186d	BAh	ø	223d	DFh	ß
150d	96h	—	187d	BBh	»	224d	E0h	à
151d	97h	—	188d	BCh	$\frac{1}{4}$	225d	E1h	á
152d	98h	~	189d	BDh	$\frac{1}{2}$	226d	E2h	â
153d	99h	™	190d	BEh	$\frac{3}{4}$	227d	E3h	ã
154d	9Ah		191d	BFh	ì	228d	E4h	ä
155d	9Bh	>	192d	C0h	À	229d	E5h	å
156d	9Ch	œ	193d	C1h	Á	230d	E6h	æ
157d	9Dh		194d	C2h	Â	231d	E7h	ç
158d	9Eh		195d	C3h	Ã	232d	E8h	è
159d	9Fh	ÿ	196d	C4h	Ä	233d	E9h	é
160d	A0h	”	197d	C5h	Å	234d	EAh	ê
161d	A1h	ı	198d	C6h	Æ	235d	EBh	ë
162d	A2h	¢	199d	C7h	Ç	236d	ECh	ì
163d	A3h	£	200d	C8h	È	237d	EDh	í
164d	A4h	¤	201d	C9h	É	238d	EEh	î
165d	A5h	¥	202d	CAh	Ê	239d	EFh	ï
166d	A6h	—	203d	CBh	Ë	240d	F0h	ð
167d	A7h	§	204d	CCh	Ì	241d	F1h	ñ
168d	A8h	¨	205d	CDh	Í	242d	F2h	ò
169d	A9h	©	206d	CEh	Î	243d	F3h	ó
170d	AAh	ª	207d	CFh	Ï	244d	F4h	ô
171d	ABh	«	208d	D0h	Ð	245d	F5h	õ
172d	ACh	¬	209d	D1h	Ñ	246d	F6h	ö
173d	ADh		210d	D2h	Ò	247d	F7h	÷
174d	AEh	®	211d	D3h	Ó	248d	F8h	ø
175d	AFh	—	212d	D4h	Ô	249d	F9h	ù
176d	B0h	°	213d	D5h	Õ	250d	FAh	ú
177d	B1h	±	214d	D6h	Ö	251d	FBh	û
178d	B2h	²	215d	D7h	×	252d	FCh	ü
179d	B3h	³	216d	D8h	Ø	253d	FDh	ý
180d	B4h	,	217d	D9h	Ù	254d	FEh	þ
						255d	FFh	ÿ

## Referencias

- [1] American National Standards Institute. Information Processing Systems Committee X3 and Computer and Business Equipment Manufacturers Association. Rationale for draft proposed American National Standard for information systems: programming language C: X3J11/88-15: Project: 381-D. Technical report, available from Global Engineering Documents, Computer and Business Equipment Manufacturers Association, 1988. 14 November 1988.
- [2] IEEE Task P754. *ANSI/IEEE 754-1985, Standard for Binary Floating-Point Arithmetic*. August 1985. Revised 1990. A preliminary draft was published in the January 1980 issue of IEEE Computer, together with several companion articles.
- [3] Brian W Kernighan and Dennis M Ritchie. *El lenguaje de programación C*. Pearson Educación, 1991.
- [4] Richard M Stallman et al. *Using and porting the GNU compiler collection*, volume 86. Free Software Foundation, 1999.