Introducción a C

¿Qué es C?

C es un lenguaje de programación que fue desarrollado en 1972 por Dennis Ritchie en AT&T Bell Labs. Se llama C simplemente porque ya existía un lenguaje de programación llamado B. Y, en realidad, el lenguaje B condujo al desarrollo de C.
C es un lenguaje de programación de alto nivel. De hecho, es uno de los lenguajes de programación de propósito general más populares (así como Java o Perl).
En computación, entre más alejado esté un lenguaje de programación de la arquitectura de la computadora, su nivel será más alto. Los lenguajes de nivel más bajo son los lenguajes de máquina que las computadoras entienden y ejecutan directamente. Por otra parte, los lenguajes de programación de alto nivel se asemejan más al lenguaje humano.
Ejemplo:

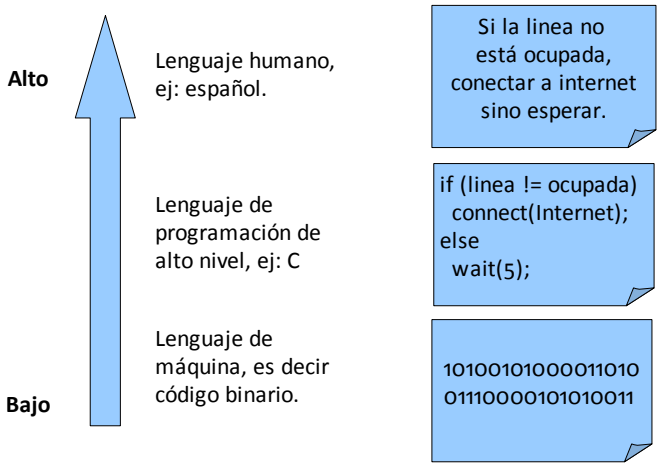

Figura 1: El espectro del lenguaje.

Los lenguajes de programación de alto nivel, incluyendo a C, tienen las siguientes ventajas:

- Legibilidad: Los programas son fáciles de leer.
- Facilidad de mantenimiento: Es fácil dar mantenimiento a los programas.
- Portabilidad: Es fácil portar los programas a través de diferentes plataformas de cómputo.

Cada lenguaje de alto nivel necesita un compilador o intérprete para traducir las instrucciones escritas en el lenguaje de programación de alto nivel a un lenguaje de máquina que la computadora pueda entender y ejecutar.
Un programa escrito en un lenguaje de alto nivel, como C, Java, Perl, C#, etc, es sólo un archivo de texto compuesto por caracteres y palabras. A estos archivos se los denomina código fuente.
Luego de que un programa es compilado, al código obtenido después de la traducción se le llama código binario. La unidad más pequeña de código binario se llama bit (binary digit), y puede tener una valor de 0 a 1. Ocho bits conforman un byte y la mitad de un byte(4 bits) es un nibble.

Un programa en C

```
/* T02CF01.cpp: Introducción a C */
#include <stdio.h>
int main(void){
 printf("¡Hola Mundo!");
 return 0;
}
```

T02CF01.cpp - Descargar

Se puede observar que la primer línea comienza con la combinación de barra + asterisco /* y termina con asterisco + barra */. En C, a la combinación /* se le denomina marca de apertura de comentario y a la combinación */, marca de cierre de comentario. El compilador de C ignora todo lo que se encuentre entre las marcas de apertura y cierre de comentario, ver Comentar el código.
Luego le sigue la directiva #include. En C, #include forma una directiva de preprocesador que indica al preprocesador de C que busque un archivo y coloque el contenido del mismo en donde la directiva #include indique.
El preprocesador es un programa que hace algunos preparativos para el compilador de C antes de compilar el código.
También puede verse que luego de #include se encuentra <stdio.h>, esto significa que es el archivo stdio.h el que debe ser buscado por el preprocesador de C.
El archivo stdio.h significaría (standard input/output header). Este archivo contiene numerosos prototipos y macros para realizar operaciones de entrada o salida.
Lo realizado en la línea 2 generalmente se llama incluir una librería, ver Archivos de encabezado.
Luego en la línea 3 se encuentra la función main. Todo programa en C debe tener una (y sólo una) función main(). Se puede colocar esta función en cualquier parte de su archivo fuente de C, pero la ejecución de su programa siempre comienza con la función main().
En este código en particular se puede ver que main tiene la palabra int a su izquierda y void entre los paréntesis de la derecha. Esta particularidad la tienen todas las funciones de C, lo que se encuentra a la izquierda indica que la función deberá devolver un valor entero (int) como se puede ver en la línea 5 y lo que está entre parentesis es el parámetro (pueden ser más de uno) que recibe la función, en este caso es void lo que indica que no se espera ningún parámetro.
Luego en la línea 4 vemos la función printf que se encuentra en la libería stdio.h, esta función en particular sirve para mostrar valores por pantalla, en este caso la cadena de texto "¡Hola Mundo!".

Comentar el código

Como ya habíamos explicado, comentar el código sirve para que el compilador ignore las líneas de código comentadas.
Esto es realmente útil para documentar el programa o para testear una parte del mismo sin necesidad de que se compilen las demás.
Si se agregan muchos comentarios a un programa, no aumenta el tamaño del archivo ejecutable ni disminuye la velocidad de rendimiento.
El lenguaje C permite dos tipos de comentarios, de una línea o de varias líneas.

```
/*
Este comentario no incrementa
el tamaño del archivo ejecutable
ni afecta la velocidad de rendimiento.
*/
```

```
//Este comentario no incrementa
//el tamaño del archivo ejecutable
//ni afecta la velocidad de rendimiento.
```

Archivos de encabezado

Los archivos que se incluyen mediante la directiva #include, como stdio.h, se denominan archivos de encabezado debido a que las directivas #include se colocan casi siempre al inicio de los programas en C. De hecho, la extensión .h significa header (encabezado), y se suele hacer referencia a ellos como archivos punto h.
Además de stdio.h, existen más archivos de encabezado, ver Tabla 1.

Paréntesis angulares (< >) y comillas dobles (" ")

En C, los paréntesis angulares solicitan al preprocesador de C que busque el archivo de encabezado en un directorio distinto al actual, generalmente llamado include. Por ejemplo, yo tengo instalado el Code::Blocks el cual está instalado en C:\Archivos de programa\ por lo que el directorio include se encuentra en C:\Archivos de programa\CodeBlocks\MinGW\include, por lo tanto, si deseo ejecutar un código que contenga #include <stdio.h>, el archivo stdio.h debe estar alojado en la ruta mencionada anteriormente.
Las comillas dobles indican al preprocesador que busque el fichero de forma relativa al directorio actual. Por lo que si su programa fuente se encuentra en C:\Programa entonces su archivo fuente deberá encontrarse también en esa ruta.
Ej:

```
#include "mi_archivo.h"
#include <stdio.h>
#include "inc/otra_libreria.h"
```

En el ejemplo anterior, mi_archivo.h debe estar en la ruta donde se encuentra el archivo fuente y otra_libreria.h debe estar en un subdirectorio llamado inc en la ruta donde se encuentra el archivo fuente.

Archivo	Descripción
assert.h	Contiene funciones de diagnóstico.
ctype.h	Contiene funciones de prueba y asociación de caracteres.
errno.h	Contiene constantes para procesar errores.
float.h	Contiene constantes para valores de punto flotante.
limits.h	Contiene valores dependientes de la implementación.
locale.h	Contiene la función setlocale() y se usa para asignar parámetros locales.
math.h	Contiene funciones matemáticas.
setjmp.h	Contiene las funciones setjmp() y longjmp(), y se usa para pasar por alto la llamada normal a una función y regresar al orden.
signal.h	Contiene funciones de manejo de señales.
stdarg.h	Contiene funciones y macros para implementar funciones que aceptan un número variable de argumentos.

stddef.h	Contiene definiciones de las macros prtdiff_t, size_t, NULL y errno.
stdio.h	Contiene funciones de entrada y salida.
stdlib.h	Contiene funciones generales de utilería.
string.h	Contiene funciones para manipular cadenas.
time.h	Contiene funciones para manipular la fecha y la hora.

Tabla 1: Archivos de encabezado del estándar ANSI.

Empezando a programar

Identificadores

Un identificador en un lenguaje de programación es un nombre utilizado para referir un valor constante,una variable, una estructura de datos compleja, o una función, dentro de un programa. Todo identificador está formado por una secuencia de letras, números y caracteres de subrayado, con la restricción de que siempre debe comenzar por una letra o un subrayado y que no puede contener espacios en blanco.
Cada compilador fija un máximo para la longitud de los identificadores, siendo habitual un máximo de 32 caracteres.
C diferencia entre mayúsculas y minúsculas, según lo cual C consideraría los identificadores contador, Contador y CONTADOR, por ejemplo, como diferentes.
Tampoco pueden utilizarse las palabras reservadas del lenguaje para la construcción del nombre de un identificador.

Palabras reservadas del lenguaje C según ANSI C

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**

Tabla 2: Palabras reservadas de ANSI C

Variables

Toda variable debe declararse antes de ser usada por primera vez en el programa. Las sentencias de declaración de variables indican al compilador que debe reservar cierto espacio en la memoria del ordenador con el fin de almacenar un dato de tipo elemental o estructurado. Por ejemplo, la siguiente declaración de variables indica al compilador que debe reservar espacio en la memoria para tres variables de tipo entero, a las que nos referiremos con los nombres a, b y c:

```
int a, b, c;
```

La declaración consiste en dar un nombre significativo a la variable e indicar el tipo de datos a que corresponden los valores que almacenaría. A continuación se muestra la sintaxis más sencilla de una sentencia de declaración para una sola variable:

```
tipo datos nombre variable;
```

Además, en una sola sentencia pueden declararse varias variables de un mismo tipo de datos, separando los nombres de las variables mediante comas:

```
tipo datos nombre_variable1,  nombre_variable2, nombre_variableN;
```

Opcionalmente, es posible asignar un valor inicial a las variables en la propia declaración:

```
tipo datos nombre variable = valor inicial;
```

Ejemplos:

```
int numero = 10;
char letra = 'b';
float num = 3.14;
```

Constantes

C admite dos tipos de constantes, las literales y las simbólicas.

Constantes literales

Todo valor que aparece en el código fuente cada vez que es necesario para una operación es denominado una constante literal.

```
Int cont = 10;
cont = cont + 5;
```

En el ejemplo anterior, 10 y 5 son constantes literales del tipo de dato entero.

Constantes simbólicas

Una constante simbólica es una constante representada mediante un nombre (símbolo) en el programa.
Al igual que las constantes literales, no pueden cambiar su valor. Sin embargo para usar el valor constante, se utiliza su nombre simbólico, de la misma forma que lo haríamos con una variable. Una constante simbólica se declara una sola vez, indicando el nombre y el valor que representa.
Las constantes simbólicas tienen dos ventajas claras respecto a las literales. Supongamos el siguiente código para calcular el perímetro de una circunferencia y el área del círculo que define:

```
perimetro = 2 * 3.14 * radio;
area = 3.14 * radio * radio;
```

Si por el contrario se hubiese definido una constante simbólica de nombre PI y valor 3.14, podríamos escribir un código mucho más claro:

```
perimetro = 2 * PI * radio;
area = PI * radio * radio;
```

Es más, imaginemos ahora que para incrementar la precisión del cálculo se desea usar un valor más preciso de la constante ∏, como 3.14159. En el primer caso debería substituirse uno a uno el valor 3.14 en todo el programa. En el segundo caso, bastaría cambiar la definición de la constante PI con el nuevo valor. Generalmente las constantes se definen de la siguiente manera:

```
const tipo_de_dato nombre_variable = valor_constante;
```

Ejemplos:

```
const int HORAS_TRABAJO = 8;
const float PI = 3.14159;
const int TRUE = 1;
const int FALSE = 0;
```

Bibliografía:

- Zhang, Tony, Aprendiendo C en 24 horas, Pearson Educación, México, 2001, ISBN: 968-444-495-8.