Fundamentos de Programación – Programación Estructurada en C:
1.- INTRODUCCIÓN AL LENGUAJE C

Maider Huarte Arrayago

Fundamentos de Programación – Programación Estructurada en C:

1.- INTRODUCCIÓN AL LENGUAJE C

1.- INTRODUCCIÓN AL LENGUAJE C

1.1.- INTRODUCCIÓN

El lenguaje de programación C se usa hoy en día en casi todas las áreas de la Ingeniería. En su día, fue presentado como un lenguaje simple y poderoso, con el que se pueden escribir todo tipo de programas; hoy en día, hay lenguajes que se usan para programar con técnicas más modernas (Programación Orientada a Objetos) que se basan en C (C++, Java).

Es un lenguaje de Alto Nivel, que requiere de un traductor de tipo Compilador para poder realizar programas ejecutables derivados de programas fuente. El proceso de compilación de un fichero fuente simple en C es el siguiente:

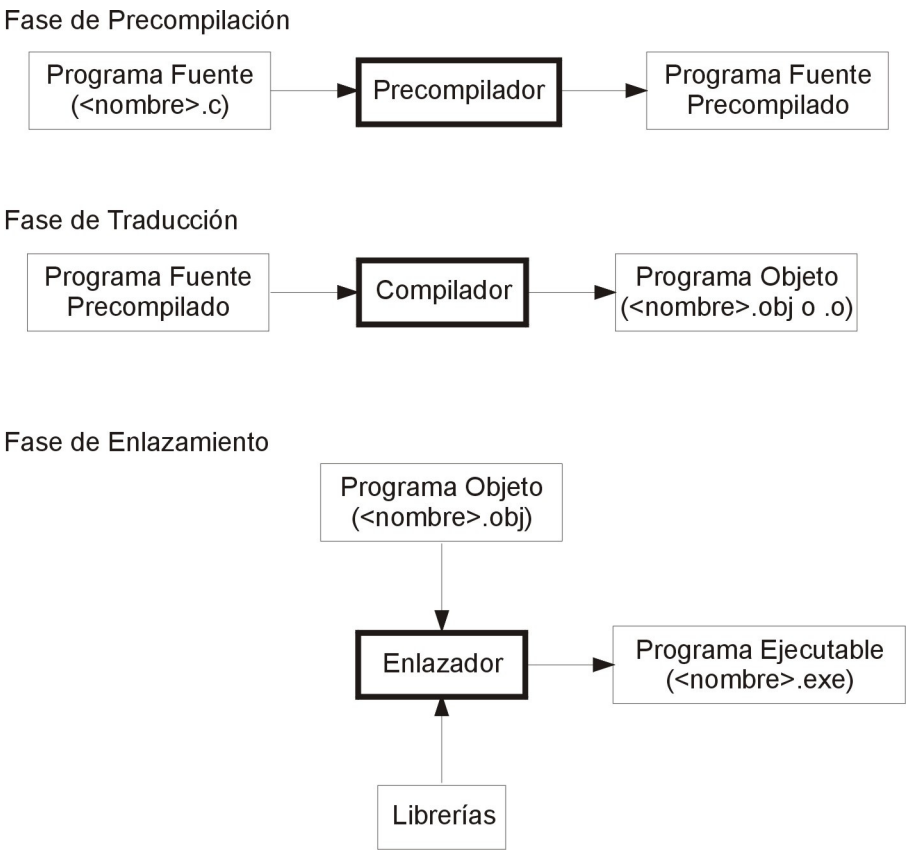

Figura 1: Compilación en C

Tal y como hemos visto en la Figura 1, la compilación de un programa fuente en C, es un proceso que, a su vez, tiene 3 procesos internos.

- **Precompilación**: Es un proceso en el que se *prepara* el fichero fuente para poder ser compilado. Esa preparación, se indica en el fichero fuente escrito por el programador, mediante instrucciones o sentencias de

precompilador, que son un tipo de instrucciones especiales del C. En el proceso de precompilación, se leen y se ejecutan solamente esas instrucciones, que se distinguen de las demás en que el primer carácter de todas es '#'.
Un fichero fuente en C, **siempre** ha de tener una extensión *.c*.
El resultado de la precompilación es un fichero fuente precompilado, que es un fichero fuente preparado para que lo pueda compilar el programa compilador.

- **Traducción**: En este proceso de un programa escrito en C, se parte de un fichero precompilado y se traduce al Código Máquina correspondiente, obteniendo un fichero objeto. Este proceso lo realiza el proceso compilador propiamente dicho.
  El fichero objeto lo habrá creado el compilador con extensión *.obj* o *.o*.
- **Enlazamiento**: En este último proceso, se enlaza el fichero objeto (obtenido como resultado de la traducción del compilador) con ficheros *librerías* (e incluso, si fuera necesario, con otros ficheros objeto), obteniendo un fichero ejecutable, con extensión *.exe*.
  Los ficheros librerías son ficheros objeto que corresponden a ficheros fuente con funciones útiles para el programador. Los programadores pueden realizar sus propias librerías de funciones, pero hay funciones que son comunes para todas las necesidades, y se recogen en librerías que forman parte de la especificación del estándar C.
  Ejemplo:

> Un ejemplo de funciones comunes recogidas en librerías, son las funciones llamadas de Entrada/Salida. Son funciones que se encargan de recoger lo que el usuario introduce por la entrada estándar del computador (en un PC, por defecto, sería el teclado) y de sacar lo que se les indique por la salida estándar (en un PC, la pantalla).
>
> Dado que casi todos los programas necesitan que su usuario introduzca datos o mostrárselos en algún momento, estas funciones están ya programadas en librerías. En los ficheros fuente, simplemente hay que indicar su uso, no hace falta escribir su código (las instrucciones que conforman esas funciones). Así, se enlazan de forma adecuada al resto de ficheros objeto para obtener el fichero ejecutable final.

Así, un software estándar de compilación para C, entre otras cosas, tiene un precompilador, un compilador, un enlazador y una serie de librerías estándar.

1.1.1.- Librerías estándar

Las librerías estándar más comunes y que más se van a usar en este curso, son las siguientes:

- **stdio**: Funciones de Entrada/Salida estándar.

- **stdlib**: Funciones de uso general. Por ejemplo, funciones para llamar al sistema operativo (p. e., para borrar la pantalla), conversión de cadenas de caracteres a números, gestión de memoria,...
- **string**: Funciones de manipulación de cadenas de caracteres.
- **math**: Funciones matemáticas. Por ejemplo, logaritmos, exponenciales, potencias de números, seno/arcoseno, coseno/arcocoseno, tangente/arcotangente,...
- **time**: Funciones para trabajar con la fecha y hora actual.

Para poder usar una función que no esté definida en el fichero fuente que se está escribiendo, sino que está en otro fichero o librería, hay que conocer primero su *prototipo*. Por eso, las librerías disponen de *ficheros de cabecera* en los que se incluyen las declaraciones de los prototipos de las funciones. El contenido de esos ficheros de cabecera se puede ver con un simple Editor de Texto, como el contenido del fichero fuente.

1.1.2.- Declaraciones

Hay que tener en cuenta, que un fichero ejecutable de C, es totalmente independiente, es decir, todo lo que la CPU necesita saber para ejecutar un fichero ejecutable en C, está codificado dentro de ese mismo fichero. Hay otros lenguajes, más modernos, que cuando un programa necesita algún recurso, pueden acceder a él, incluso a través de Internet, pero no es el caso de C.

Aparte de las estructuras sintácticas y palabras reservadas propias del lenguaje, **el compilador no admite la utilización de nada que antes no haya sido *indicado***, y da **error** (con lo que no se genera ningún fichero objeto ni se puede crear ningún ejecutable). Es decir, lo único conocido de antemano por el compilador, son las estructuras sintácticas y las palabras reservadas.

Eso no significa que el programador no pueda introducir elementos nuevos en un programa; todo lo contrario, la programación se basa en poder usar elementos nuevos según vea el programador que son necesarios. Lo único que hay que hacer para que el compilador no de ningún error, es indicar al compilador que va a haber un elemento nuevo en el fichero, antes de usar ese elemento. La forma de hacer esa indicación al compilador, se conoce como **declaración** de un elemento.

Dependiendo del tipo de elemento del que se trate, la declaración se hace de forma distinta, pero en toda declaración, se indica el **nombre** del nuevo elemento que se está declarando. Los nombres o **identificadores** elegidos, han de seguir unas normas:

- Sólo se pueden usar caracteres alfabéticos (‘A’,..., ‘Z’; ‘a’,..., ‘z’), numéricos (‘0’,..., ‘9’) y el carácter imprimible ‘_’.

- Hay que tener en cuenta que para el compilador las letras mayúsculas y minúsculas son caracteres diferentes (tienen diferente código ASCII), con lo cual, no se pueden usar para el mismo elemento en un programa, identificadores que usen las mismas letras y en el mismo orden, pero cambiando mayúsculas por minúsculas o al revés.

  Ejemplo:

  > Los siguientes son identificadores diferentes que se pueden usar para diferentes elementos:
  >
  > Nombre_dato, NOMBRE_DATO, nombre_dato, nOMbre_DATo

- El primer carácter del identificador, no puede ser numérico, sólo puede ser alfabético o '_'.

1.1.3.- Estructura general de un programa en C

Un programa fuente en C, como cualquier Lenguaje de Programación Estructurada de Alto Nivel, tendrá una estructura general que habrá que tener en cuenta a la hora de redactarlo. Esa estructura general, se encaja en la vista en el tema **2.- PROCESO DE CREACIÓN DE UN PROGRAMA** (**2.3.- DISEÑO DE ALTO NIVEL DEL PROGRAMA**) de la anterior parte de la asignatura (**Introducción a la Programación y Lenguajes de Programación**), de forma genérica:

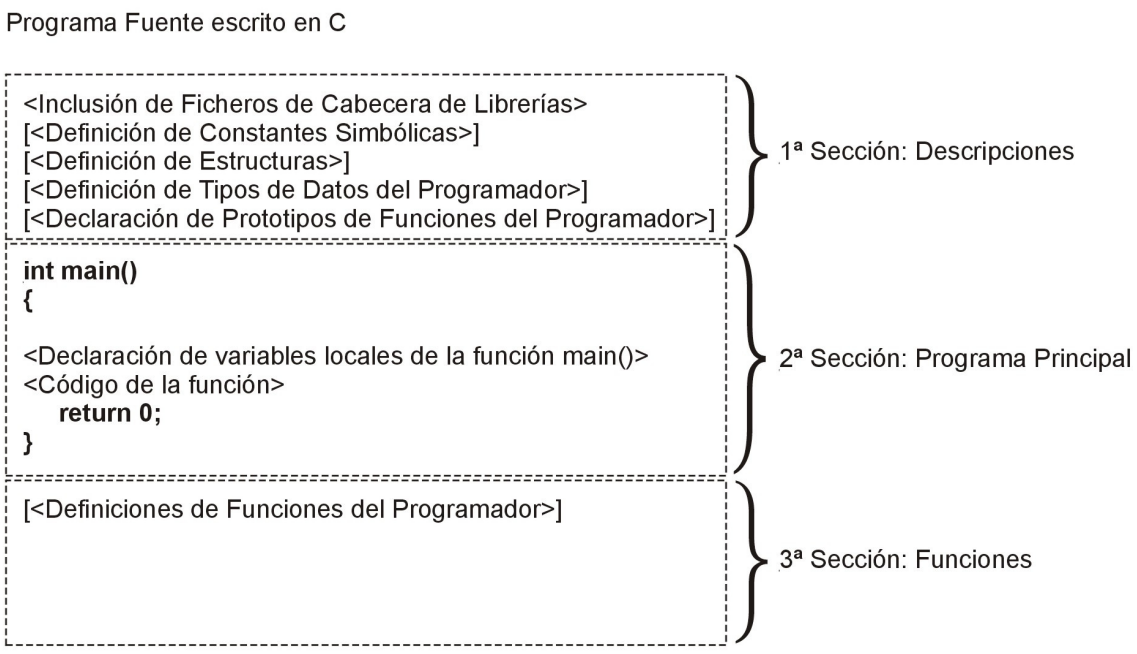

Figura 2: Estructura general de un programa en C

Tal y como se ve en la figura, se sigue una estructura con 3 secciones:

- **1ª Sección: Descripciones**

  La Inclusión de Ficheros de Cabecera de Librerías, Definición de Constantes Simbólicas y Definición de Tipos de Datos del Programador, se hacen mediante instrucciones del precompilador, siendo las dos últimas, opcionales. Se verán en un tema posterior.
  La Definición de Estructuras y Declaración de Prototipos de Funciones son también opcionales. También se verán en un tema posterior.
  Es importante seguir el orden indicado al escribir el contenido de esta sección.

- **2ª Sección: Programa Principal**

  Es el punto donde la CPU encontrará la 1ª instrucción ejecutable de todo el programa. Para indicar el comienzo de esta sección, es necesario escribir su encabezamiento y su contenido como un bloque entre llaves; en máquinas con sistema operativo tipo Linux, el encabezamiento ha de ser *int main()*, y la última línea dentro del bloque, *return 0;*. En el bloque de esta sección estarán las instrucciones que implementan el algoritmo de 1er nivel de resolución del problema.

- **3ª Sección: Funciones**

  En esta sección se escriben las definiciones de las funciones que el programador ha inventado y usado en el Programa Principal. Si no hubiera ninguna, esta sección estaría vacía. Se verá en el tema posterior correspondiente. En cada función de esta sección estarían las instrucciones que implementan los demás algoritmos de resolución del problema.

1.2.- TIPOS DE DATOS

El lenguaje C puede trabajar con un amplio conjunto de tipos de datos. Cada tipo de dato tiene un nombre concreto, que será una palabra reservada que sólo puede usarse para identificar a un cierto tipo. La diferencia entre los tipos de datos, aparte de su contenido conceptual, está en la cantidad de bytes que ocupan en memoria; dependiendo de esos dos factores, el rango de cada tipo de dato es diferente.

La siguiente tabla ilustra los diferentes tipos de datos en C:

TIPO DE DATO	NOMBRE	Nº DE BYTES	RANGOS
Carácter	**char**	1	[-128, 127]
Números Enteros	**unsigned short**	2	[0, 65.535]
	short	2	[-32.768, 32.767]
	unsigned int	4	$[0,2^{32}]$
	int	4	[-2.147.438.648, 2.147.438.648]
Números Reales	**float**	4	$[-3.4x10^{38}, -1.18x10^{-38}]$ 0 $[1.18x10^{-38}, 3.4x10^{38}]$
	double	8	$[-1.79x10^{308}, -2.23x10^{--308}]$ 0 $[2.23x10^{--308}, 1.79x10^{308}]$

Tal y como vimos en el punto 1.2.1.1.- Memoria del tema **1.- LENGUAJES DE PROGRAMACIÓN**, todos los caracteres imprimibles están tabulados de forma que cada uno tiene asignado un número que en binario, se representa mediante un byte. Por eso, cuando en memoria se guarda un carácter (un valor de tipo *char*) lo que se guarda realmente es un número que ocupa un byte.

En el caso de C, la codificación numérica que se usa con los caracteres es la Tabla ASCII. De esta tabla, lo único que es **imprescindible** saber, es que los caracteres alfanuméricos ('a',..., 'z'; 'A',..., 'Z'; '0',..., '9') están ordenados, y que el rango de esos códigos va desde -128 a 127.

1.3.- SINTAXIS DE C

Aparte de lo que se explica en este apartado, la sintaxis de C también abarca un conjunto de operadores y de sentencias de control, que se verán en otros temas posteriores.

En un fichero fuente escrito en lenguaje C, puede haber 4 clases de líneas:

- **Instrucciones del precompilador**: Son instrucciones que no se ejecutan en el proceso de ejecución del programa sino que habrán sido leídas y ejecutadas por el proceso precompilador. Así, el fichero fuente precompilado que se compila, no las tendrá como tales, sólo tendrá el resultado de su ejecución en el precompilador.

- **Comentarios**: Son líneas de código que sólo tienen significado para los programadores; suelen ser notas aclaratorias o recordatorias. El compilador las ignora, no las compila, porque no se ejecutan. En C, hay dos formas de escribir comentarios:
  - **Comentario de una sola línea**: Se indica anteponiendo al comentario "//".
  - **Comentario de más de una línea**: Se indica poniendo "/*" antes de la primera línea del comentario, y "*/" después de la última línea.

Ejemplo:

```
// Esto es un comentario de una sola línea

/* Y esto
es otro comentario,
de más de una línea */
```

- **Sentencias**: Son líneas en las que se indica una instrucción, que se compila y es ejecutada por el computador en tiempo de ejecución. Termina con un ";".

- **Llaves de apertura y cierre**: Se trata de los símbolos '{' (llave de apertura) y '}' (llave de cierre). Sirven para delimitar bloques de sentencias, que han de ejecutarse como un todo. Es decir, cuando el computador empieza a ejecutar las sentencias de un bloque, no puede ejecutar una instrucción de fuera del bloque sin antes terminar de ejecutar todas las sentencias internas del bloque.

La notación que se va a usar a partir de este punto, para indicar una estructura sintáctica concreta, va a ser la siguiente:

- Todo lo que se escriba en **negrita**, es literal, forma parte de la estructura sintáctica que se está explicando, por lo que cada vez que se use esa estructura, ha de aparecer todo lo indicado en negrita.

- Todo lo escrito entre '<' y '>', hay que interpretarlo, y sustituir todo el conjunto ('<' y '>' incluidos) por lo que se indica en lo escrito.

  Ejemplo:

  ```
  <nombre>
  <tipo>
  ```

- Todo lo escrito entre '[' y ']' es opcional. Es decir, no es necesario para que la estructura sintáctica esté correcta.

- El signo '[...]' indica que se puede seguir poniendo lo que se ha puesto anteriormente.

1.3.1.- Variables y Constantes

Los datos que maneja un programa, se pueden guardar de dos formas, como variables o como constantes. Cuando el valor de un dato cambia a lo largo de un programa, se guarda en una variable, y si no lo hace, es mejor guardarla en una constante.

1.3.1.1. Variables

Desde el punto de vista de la CPU, una variable es una zona de memoria cuyo valor va a ir cambiando durante la ejecución de un programa.

Las variables usadas en un programa se ajustan a las necesidades del problema que se intenta resolver mediante el mismo. Es decir, las variables son elementos nuevos para el compilador, por lo cual, según lo explicado en el punto **1.1.- INTRODUCCIÓN**, es necesario **declararlas**.

Declaración:

<tipo> <identificador>**;**

Así, se declara una variable. Se indica primero qué tipo de dato va a representar esa variable, y después, cuál va a ser su identificador (su nombre).

Se puede aprovechar y declarar más de una variable a la vez, pero, por supuesto, han de ser variables del mismo tipo. La sintaxis es la misma, sólo que se separan los identificadores de las variables por ‘,’:

<tipo> <identificador de la 1ª variable>**,**<identificador de la 2ª variable>[...]**;**

Tal como se ve en la sintaxis de la declaración, el ‘**;**’ es necesario. Eso indica que la declaración se trata de una sentencia (**1.3.- SINTAXIS DE C**). En el caso de la declaración de variables, lo que se hace es una **reserva de memoria**. Cuando la CPU encuentra el código de la **declaración de una variable**, lo que hace es **buscar en** la **Memoria Principal**, los **bytes seguidos suficientes** como para contener el tipo de dato que guarda la variable, **y reservarlos**. La CPU, siempre lleva el control de las zonas de memoria que tiene reservadas y con qué proceso las tiene reservadas, mediante tablas (que, a su vez, como todo lo que hace la CPU estarán en otra zona de memoria,...).

Hay que tener en cuenta, que al hacer la reserva de memoria, los bytes reservados siempre contendrán algún valor (porque han sido usados anteriormente con alguna otra variable ya desaparecida, por ejemplo). Por eso, **si no queremos que una variable tenga un valor desconocido al principio**, hay que **inicializar** la variable **con un valor**, **en** el mismo momento de **la declaración**. Para ello, se usa el **operador asignación**:

<tipo> <identificador>**=**<valor>**;**

Así, después de hacer la reserva de bytes, el valor indicado a la derecha del operador ‘=’, se introduce en la variable indicada a su izquierda.

Ejemplo:

```
int x,y,z=0;
```

Variable	Dirección	Memoria	Tamaño
x	1000 ... 1003	-	4 bytes
y	1100 ... 1103	-	4 bytes
z	1200 ... 1203	0	4 bytes

Las variables *x* e *y* tiene valores desconocidos, porque no han sido inicializadas a ningún valor en concreto. La variable *z* sin embargo, ha sido inicializada a 0, y ese es el valor que se indica en los bytes de la misma.

En general, se puede dar valor a una variable mediante el operador asignación, indicando a la derecha del mismo, otra variable, un valor constante, o una expresión (conjunto de operaciones sobre variables y/o constantes). En el ejemplo anterior, se ha visto el caso en el que se da valor mediante una constante.

El **ámbito** de una variable es la porción de programa donde dicha variable es visible para el código del programa, es decir, puede ser usada. El ámbito de una variable depende del lugar del programa donde es declarada, pudiendo pertenecer categorías distintas. En este curso, sólo se usarán dos de las posibles en C:

- **Variable local**: Una variable local se declara dentro del cuerpo de una **función** y es visible **únicamente** dentro de esa función. Todas las variables locales que se van a usar en una función en C, hay que declararlas al inicio de la misma, antes de escribir ningún otro tipo de sentencia.

- **Parámetro de una función**: Se verán en el tema dedicado a las funciones.

1.3.1.2.- Constantes

Son elementos que, una vez compilado el programa no pueden ser cambiados, y por lo tanto, mantienen el mismo valor en todo el proceso de ejecución del programa.

En C, hay 4 tipos de constantes o literales:

- **Constantes enteras**: Se indican mediante un número entero (sin '.'), en formato decimal, octal o hexadecimal. Siempre se consideran como de tipo *int*.

  Ejemplo:

  ```
  40
  050
  0x28
  ```

- **Constantes reales**: Se indican mediante un número real, en formato decimal (con '.') o exponencial (con 'e'). Se consideran como de tipo *double*, a no ser que se indique mediante una 'f' o 'F', que es *float*.

  Ejemplo:

  ```
  1.
  1e0
  1.f
  1.F
  1e0f
  1e0F
  ```

- **Constantes caracteres**: Se indican como un carácter entre comillas simples ('). Todos los caracteres que aparecen en la tabla ASCII, hasta los no imprimibles, se pueden representar mediante literales. En el caso de los caracteres no imprimibles, se usan *secuencias de escape*:

  La siguiente tabla muestra las secuencias de escape de los caracteres no imprimibles más usados en C:

SECUENCIA	SIGNIFICADO
'\n'	Siguiente línea
'\t'	Tabulación horizontal
'\r'	Tecla *return*
'\\'	Contrabarra
'\''	Comilla simple
'\"'	Comilla doble
'\0'	Null
'\b'	Mover el cursor un carácter hacia atrás

- **Constantes cadenas**: Son constantes o literales formados por una cadena de uno o más caracteres, y se escriben entre comillas dobles ("). Pueden usarse también las secuencias de escape indicadas en la tabla anterior.

  Ejemplo:

> Si hacemos que un programa saque por pantalla la cadena “cad\bena”, lo que saldrá en pantalla será lo siguiente: caena

En general, cuando una misma constante se usa más de una vez en un programa, es malo escribirlo de forma literal, ya que si hay algún error en ese literal, hay que cambiarlo en todas las veces en las que se ha escrito.

En vez de usar literales, se puede optar por usar **constantes simbólicas**. Eso se hace mediante instrucciones al precompilador, que se verán en el siguiente tema.