Apéndice

A

El compilador de C

En este apéndice se verá el funcionamiento del compilador de C en ambientes Windows y fundamentalmente GNU-Linux, que es el que se recomienda.

Introducción

Existe una secuencia de trabajo para producir una aplicación ejecutable desde un código fuente en ANSI C (o cualquier otro lenguaje no interpretado). El código en lenguaje de alto nivel debe ser traducido a código máquina que pueda ser ejecutado por el computador. Aquí es donde interviene el compilador, que es ayudado por varias herramientas para su cometido: el ensamblador, el enlazador y el depurador.

Las fases para producir la aplicación las podríamos resumir en los siguientes puntos:

1. Con la ayuda de un editor escribiremos un programa en alto nivel. Este editor puede ser muy sencillo y de propósito general o especializado en la realización de código fuente, con lo cual será sensible a las palabras propias del lenguaje de alto nivel utilizado, avisarnos de la falta de algún elemento como llaves o puntos y coma e incluso nos permitirá embellecer ese código. En cualquier caso el resultado será un código en texto plano.

2. Este código pasa un por una fase de pre procesamiento del texto (ver capítulo 6), que incluirá la sustitución de determinadas constantes o la inclusión de otros ficheros, por citar los dos cometidos más usuales. En cualquier caso se produce otro fichero de texto plano (modificando el anterior) con formato de código de alto nivel.

3. Este nuevo código ahora es compilado (internamente se pasa por varias fases que puedes ver en el apéndice B). Y se producirá un código en ensamblador normalmente. Puede ocurrir que el código fuente esté mal realizado, esto nos lo advertirá el compilador, indicando el tipo de fallo y la línea de código fuente que falla (ojo algunas veces puede fallar). Si falla volveremos a editar el código hasta corregirlo.

4. Una fase que puede ser opcional es el ensamblado del código producido. Si el compilador produce ensamblador (lenguaje máquina con formato humano), este ensamblador pasará por otra herramienta que es el ensamblador (ver apéndice C para más detalles), que produce código objeto en lenguaje máquina. Si el compilador hace ya esta labor esta fase no será necesaria.

5. Este código objeto aún no es ejecutable, ya que al realizar el código fuente, hemos utilizado algunas funciones de algunas librerías que ya estaban realizadas y que no hemos programado e incluido. Por ello hace falta otra fase que es la de enlazado. Nuestro código objeto se tiene que "mezclar" con otro código (el de las librerías) para por fin obtener el código ejecutable. Este trabajo es realizado por otra herramienta que se llama enlazador ("linker").

6. Llegamos a la fase de ejecución y depuración, para lo cual daremos nuestra aplicación al sistema operativo para la ejecute. En muy raras ocasiones el código que hemos hecho funciona a la primera, tanto en la fase de compilación como de ejecución, por ello existe otra herramienta llamada depurador para poder ejecutar la aplicación de forma controlada y nos que nos dé detalles de donde falla. Funciones típicas de una herramienta de depuración será la ejecución paso a paso o la visualización del valor de las variables entre otras muchas.

Las fases descritas se conocen como parte de las fases del ciclo de vida del software y su coste en términos generales aparece en la siguiente tabla:

Ciclo	Porcentaje de esfuerzo
Definición del programa	3%
Especificación	15%
Codificación y testeo*	14%
Verificación	8%
Mantenimiento	60%

Puede ser sorprendente el poco esfuerzo relativo que supone la codificación y el enorme esfuerzo que se dedica al mantenimiento, por eso una de las leyes "no escritas" de la programación es que un programa se escribe una vez y se lee cientos, por eso hay que hacer énfasis en la documentación del programa y la claridad del mismo (nombres de variables, profusión de comentarios, código claro, prototipos de funciones, etc.).

Otras de los puntos a tener en cuenta es que cuesta tanto esfuerzo codificar un programa para resolver un problema como su definición y especificación, es decir, antes de empezar a programar hay que dedicar un esfuerzo equivalente a saber qué es lo que vamos a programar. Lo podemos resumir en los siguientes pasos:

1. Tener claro el problema a resolver. Esto parece obvio.
2. Saber cuáles van a ser las entradas del programa y sus salidas.
3. Resolver el problema a mano con ejemplos cortos.
4. Desarrollar un algoritmo (pseudocódigo) para resolver el problema.
5. Codificar el algoritmo en el lenguaje escogido (ANSI C) con explicación en comentarios. Estos comentarios pueden incluso, a parte de la explicación, contener el nombre del autor, la fecha de inicio y los cambios realizados, o todo lo que se crea oportuno. Nunca ser cicatero con las explicaciones. Dentro de unos meses ni el propio autor se puede acordar de lo que ha hecho.
6. Verificación que el resultado del programa es el del ejemplo corto a mano.

Compilación en Windows

En Windows existen varias posibilidades de utilizar un compilador de C, algunas de ellas son de pago, como Visual C Studio o similares (salvo licencias especiales como la de la Universidad de Cantabria - https://sdei.unican.es/Paginas/servicios/software/Software-para-estudiantes.aspx-) o son demasiado complejas para nuestro propósito, por ello la opción que se ha escogido es la de usar un compilador de C++ (perfectamente válido para C) gratuito de software libre llamado Bloodshed Dev

C++ (http://prdownloads.sourceforge.net/dev-cpp/devcpp-4.9.9.2_setup.exe) para aquellos que prefiráis usar el entorno Windows.

Tiene un entorno de desarrollo completo (editor, compilador, depurador, ejecución en línea) como se puede ver en la Ilustración 1:

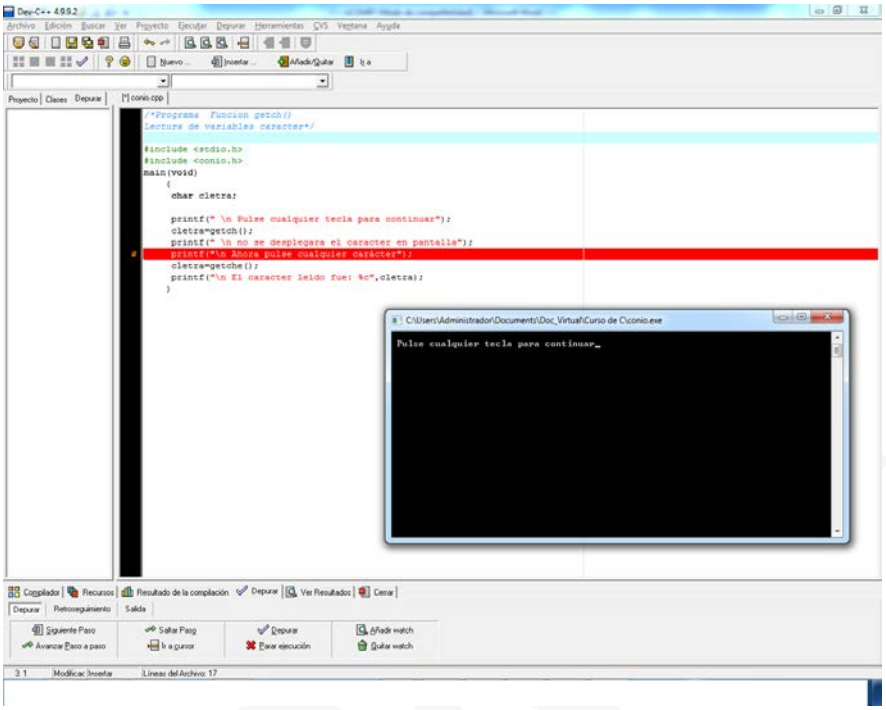

Ilustración 1. Compilador Dev C++.

Otra posibilidad es Geany (http://download.geany.org/geany-1.23.1_setup.exe) un entorno integrado de desarrollo (IDE) también de software libre que existe tanto para Windows como para Linux con similares características (Ilustración 2):

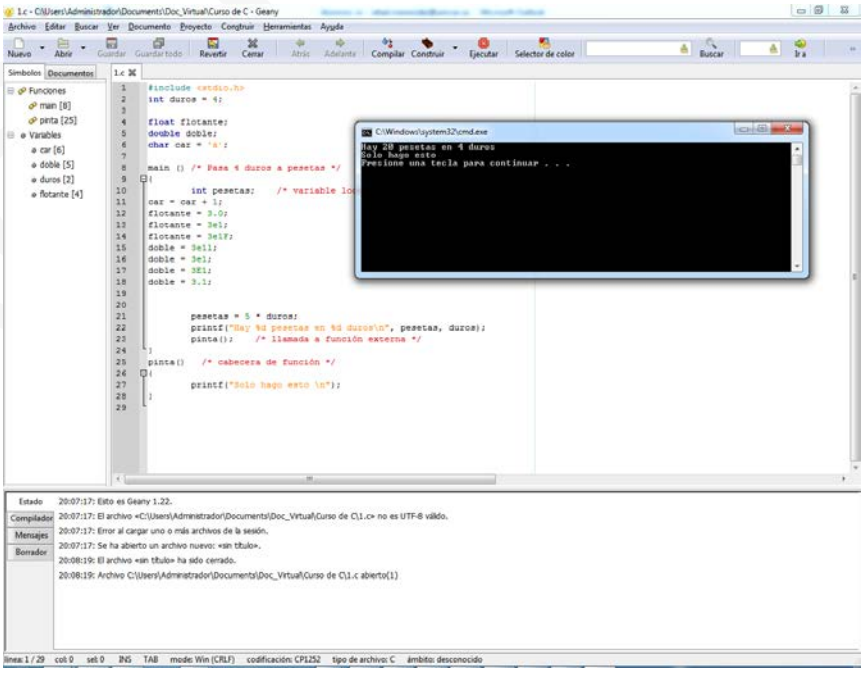

Ilustración 2. Compilador Geany.

Compilación en GNU-Linux

La otra posibilidad, que es la recomendada, es que utilicéis un entorno GNU-Linux. En este entorno también tenéis varias posibilidades que podemos clasificar en dos clases: IDE o las herramientas clásicas (separadas) de un sistema GNU-Linux.

En el primer caso podéis utilizar, dependiendo del sistema de ventanas que uséis, Kdevelop para KDE, o gedit (con plugins) para Gnome, o cualquiera de los IDE existentes como el mencionado Geany, Anjuta (http://projects.gnome.org/anjuta/downloads.html), NetBeans (http://netbeans.org/) de Sun o el archiconocido Eclipse con plugins (http://www.eclipse.org/).

En el segundo caso el compilador de C que vamos a utilizar es el gcc (g++). Este compilador realiza automáticamente toda la cadena de operaciones para producir un fichero ejecutable, es decir, llama al preprocesador, compila el programa, produce un fichero en ensamblador, ensambla el programa, lo enlaza (linka) y produce el ejecutable (cpp es el preprocesador, ld es el linker, as es el ensamblador y gdb el depurador –ddd es su versión gráfica–).

El compilador siempre espera que se le dé un nombre de fichero que contenga un programa fuente de lenguaje C, para distinguir estos ficheros de código fuente del resto de ficheros del sistema, es obligatorio que todos terminen en ".c". Si no hacemos esto el compilador nos responderá con: "file not recognized: File format not recognized". Otras terminaciones comunes son: .C, .cc, y .cxx para ficheros en c++; .h para ficheros del preprocesador; .s, .S para ficheros en ensamblador; y .o para ficheros objeto (sin enlazar).

Como cualquier otro comando del sistema, el formato para ejecutarlo será nombre [-opciones] y argumentos. La forma más sencilla de ejecutarlo para producir un fichero ejecutable con nombre a.out sería la siguiente:

```
gcc programa.c
```

Como hemos dicho, el compilador es un comando más, por lo que también admite opciones, éstas usualmente se pondrán delante del nombre del fichero fuente y deberán estar separadas, por ejemplo no es lo mismo poner gcc -dv que gcc -d –v, dado que hay opciones multiletra. Las opciones están divididas en varios grupos: globales, del lenguaje, de alerta, de depuración, de optimización, de preprocesado, de ensamblador o linkador, de directorios, y de dependencias del hardware.

A continuación aparecen las opciones más habituales (las puedes encontrar con `man gcc`):

Globales:

-c	Compila pero no llama al linker, generando un .o.
-o ejecutable	Cambia el nombre del fichero ejecutable (por defecto a.out) al expresado en la opción.

Del lenguaje:

-ansi	Compila siguiendo las reglas del ANSI C.
-E	Ejecuta sólo el preprocesador.

Del linker:

-llibreria	Incluye la librería libreria.

De directorios:

-Ldirectorio	Añade el directorio directorio a los directorios donde se buscará lo dado por -l (anterior opción).
-Idirectorio	Añade el directorio a la lista de directorios de includes.

De alerta:

-w	Inhibe los mensaje de warning. Los mensajes de warning indicarán algo a tener en cuenta, pero que no es un error.

De depuración:

-g	Produce información para poder utilizar alguno de los depuradores (debuggers) del sistema (gdb).

Una vez que se ha compilado un programa con la opción -g, puede ser ejecutado con el depurador (*debugger*) del sistema, que en este caso se llama gdb:

```
$gdb nombre_ejecutable
```

Es altamente recomendable usar la versión gráfica "ddd" (Ilustración 3) que básicamente utiliza por nosotros de forma amigable el gdb. Esto permitirá ejecutarle de forma controlada (paso a paso, línea a línea, hasta un punto de ruptura, etc.), ver los valores de las variables, etc. Para más información se puede utilizar el comando help del mismo. También se puede utilizar el depurador para obtener información de los ficheros *cores* del sistema. Cuando un programa se aborta en ejecución se produce un fichero de nombre *core* que puede ser analizado con *gdb core*.

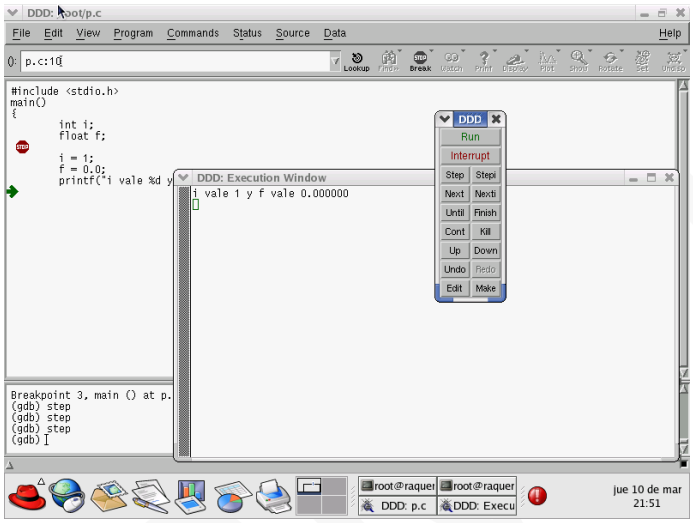

Ilustración 3. Depurador gdb usando ddd.

Compilación en entornos mixtos

Hay una tercera forma de compilar programas en C desde Windows pero en un entorno Linux que es utilizando el emulador cygwin (https://cygwin.com/index.html). Para ello debemos ejecutar el paquete instalador de 32 ó 64 bits en nuestro ordenador. Al hacerlo nos pedirá la forma de instalación, que se puede hacer directamente desde internet (hay que escoger un repositorio desde donde bajar los ficheros de instalación, se recomienda no usar el español de rediris ya que lamentablemente no funciona).

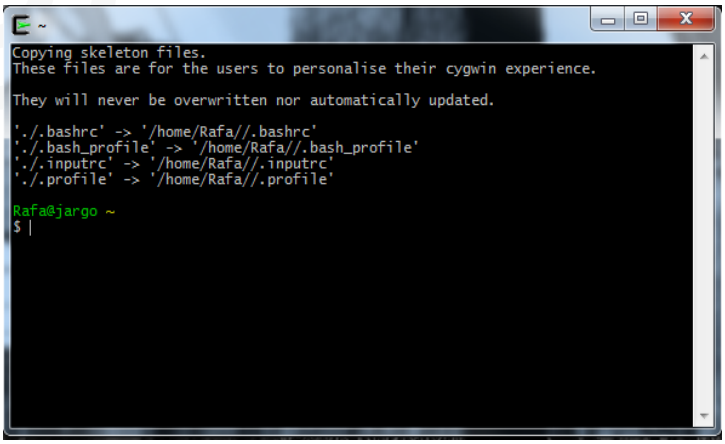

Ilustración 4. Shell en el entorno cygwin.

Una vez instalado lo ejecutaremos y nos aparecerá una ventana de Windows donde existe más o menos el mismo entorno de programación que en un terminal de Linux ejecutando una Shell (Ilustración 4). Esto significa que dispondremos de todos los comandos habituales de la Shell, como por ejemplo el editor `vi` (vim), que se puede revisar en el apéndice D, el compilador `gcc` (que producirá por defecto ficheros "`a.exe`") y además tenemos reflejo del sistema de ficheros Linux en un directorio de Windows (aquel que hemos indicado en la instalación). Y podemos pasar ficheros directamente desde Windows (Ilustración 5).

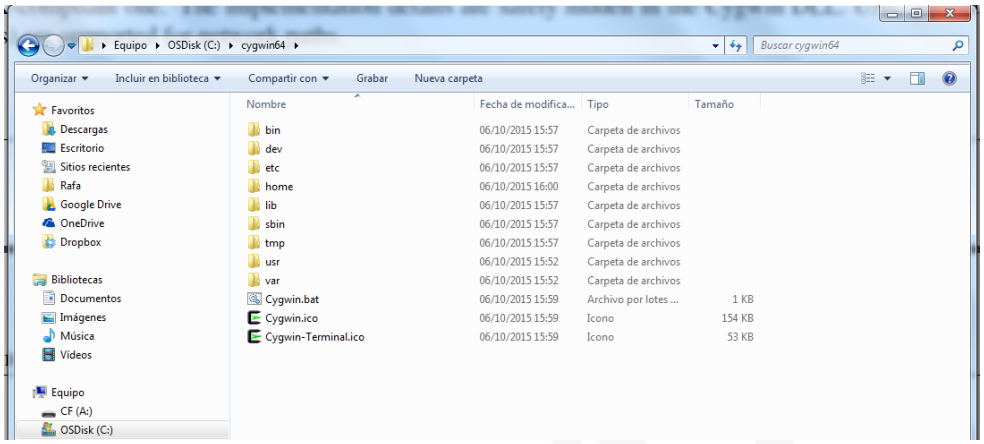

Ilustración 5. Sistema de ficheros de cygwin.

Si por alguna causa (normalmente la elección del repositorio) no estuvieran las herramientas de desarrollo (editores, compilador gcc, depurador, etc.), siempre en el proceso de instalación podremos decir que paquetes queremos que se nos instalen.

Existen otras formas mixtas de tener un entorno de múltiples sistemas operativos:

1. Con el programa wubi de Ubuntu (https://wiki.ubuntu.com/WubiGuide). Era un emulador de un sistema Linux Ubuntu en una ventana Windows (de forma análoga a cygwin) pero que además incluía toda la parte gráfica. Está en desuso.

2. Creando una máquina virtual desde Windows. Existen diversos programas gratuitos para hacer esto como:
   a. vmware player (https://www.vmware.com/es/products/player/) de VMplayer.
   b. virtual box (https://www.virtualbox.org/) de Oracle.
   c. virtual PC de Microsoft. Ya obsoleto, de 2007.

3. Creando un sistema de doble arranque. Es el más avanzado y arriesgado, exige tener varias particiones en el sistema.