Apéndice

D

Otros aspectos de C y del sistema

Orden de precedencia de los operadores

En la tabla que aparece a continuación están todos los operadores de C y sus órdenes de precedencia con más prioridad los de arriba:

Operadores	Significado	Sentido
() {} -> .	Paréntesis, Bloque, Campos de estructuras	Izda - Dcha
! ~ ++ -- - (tipo) * & sizeof	NOT, Negación bit, Incremento, Decremento, signo menos, conversión tipo, indirección, dirección, tamaño	Dcha - Izda Todos Unarios
* / %	Multiplicación, División, Módulo	Izda - Dcha
+ -	Adición, Sustracción	Izda - Dcha
>> <<	Desplazamiento de bits	Izda - Dcha
> < >= <=	Relacionales	Izda - Dcha
== !=	Relacionales	Izda - Dcha
&	AND bit	Izda - Dcha
^	XOR bit	Izda - Dcha
|	OR bit	Izda - Dcha
&&	AND	Izda - Dcha
||	OR	Izda - Dcha
?:	Condicional	Izda - Dcha
= += -= *= /= %=	Asignación	Dcha - Izda
,	Coma	Izda - Dcha

Declaraciones complejas

La combinación de los modificadores de declaración de punteros "*****", arrays "**[ ]**", y funciones "**( )**", permite en lenguaje C declaraciones muy sofisticadas pero poco claras y en principio difíciles de interpretar.

Cuando se combinan estos modificadores hay que tener presente ciertas reglas de ordenamiento:

1. La prioridad del modificador es mayor cuanto más próxima esté al identificador.
2. Los modificadores **[ ]** y **( )** tienen mayor prioridad que el *****.
3. Se pueden usar paréntesis, no confundir con las funciones, para agrupar parte de una declaración.

En los siguientes ejemplos se ven declaraciones de datos utilizando estas normas:

Declaración	Significado
char matriz[4][5]	Un array doble de 4 x 5 (4 strings de 5 char)
char *string	Un puntero a char (un string)
int **puntero	Un puntero a puntero de int
char *array[12]	Un array de 12 punteros a char (strings)
int (*pun)[3]	Un puntero a un array de 3 enteros
int *arr[4][3]	Un array de 4 punteros a arrays de 3 enteros
int (*pun)[3][5]	Un puntero a una array doble de 3 x 5

En estas declaraciones también se pueden incluir las funciones, con lo cual las expresiones se hacen aún más complicadas:

Declaración	Significado
char *funcion()	Una función de tipo puntero a char (string)
long (*punf)()	Un puntero a una función que devuelve un long
int *funci()[3]	Una función que devuelve una puntero a array de 3 enteros
char *fun[3]()	Array de tres punteros a función que devuelve un dato de tipo char

Construcción de un programa sobre GNU/Linux - UNIX

Programación separada (modular)

Hasta ahora hemos visto que todos nuestros programas estaban en un fichero que se edita, compila y ejecuta. Pero normalmente, en programas grandes, esto no se hace así, sino que se construye el programa de forma modular en varios ficheros, aplicándose el principio de *divide y vencerás*, ya que los módulos del programa serán más fáciles de entender y depurar (algo parecido a la división de un programa en funciones pero a otro nivel más abstracto). Con esta forma de trabajar conseguimos algunas ventajas:

1. Obviamente los módulos tienen una extensión menor que el programa completo. Por lo tanto, éstos serán más fáciles de analizar.
2. Cada módulo se puede compilar por separado (más rápido).
3. Cada módulo será más fácil de depurar por separado, ya que no se tendrán que tener en cuenta influencias externas.
4. La división del trabajo entre varios programadores es más sencilla y limpia de realizar.

Esto conlleva que se tengan que aplicar (conveniente no obligatorio) ciertos criterios a la hora de construir ese programa utilizando la estructura en árbol de directorios y ficheros:

1. Se puede utilizar un directorio (en vez de un fichero como antes) para contener los ficheros de los que va a estar constituido el programa.

2. Dentro de ese directorio general se puede crear varios subdirectorios donde se sepa que se va a encontrar lo que estamos buscando, como por ejemplo:
    2.1. Un directorio para los ficheros fuente.
    2.2. Un directorio para los ficheros de cabecera del preprocesado (normalmente "include"). Ver punto 4. Los del sistema están en el directorio "/usr/include".
    2.3. Un directorio de librerías (normalmente "lib").
    2.4. Un directorio de ejecutables (normalmente "bin").
    2.5. Un directorio de documentación (normalmente "doc").

3. Los módulos tienen que ser construidos (división del programa) teniendo en cuenta principios semánticos, ofreciendo servicios a otros módulos externos (cajas negras), de tal manera que se garantice el perfecto funcionamiento de los mismos de forma aislada. También se deberá tener en cuenta que[1]:
    3.1. Hay partes dependientes del hardware que deben ser señaladas como tal. De hecho, por definición no son transportables a otros sistemas y deben estar separadas del resto del programa.
    3.2. Otras dependerán de algo específico como llamadas a un sistema operativo concreto y deberán ser tratadas de la misma forma.

4. Los ficheros de cabecera "*.h" están destinados normalmente a contener las definiciones comunes a varios módulos. En ellos suelen aparecer distintos tipos de información:
    4.1. Definición de constantes.
    4.2. Definición de tipos de datos.
    4.3. Definición de prototipos de funciones.

   Suele ser una mala práctica de programación incluir las propias definiciones (reservas de espacio) de variables.

Gestión de librerías en UNIX

Anteriormente se ha comentado que podemos incluir en nuestro programa, código objeto realizado en otros lenguajes o en el mismo C, así la línea de compilación podría complicarse:

```
gcc modulo1.c modulo2.c prepro.i programa.c objeto.o -o eje -lm
```

donde hemos incluido al compilar dos módulos de código fuente C, un fichero de preprocesado, el programa principal fuente, un código objeto (no necesariamente C) y una librería, en este caso la librería matemática (para ello deberemos haber usado en alguna parte un *#include* de *math.h*).

Una pregunta que podemos hacernos es por qué hemos incluido esa librería. La respuesta es porque con el fichero de cabecera *math.h* sólo hemos incluido definiciones de constantes, tipos de datos y prototipos de funciones, pero no el cuerpo compilado de estas funciones, que está contenido precisamente en esa librería (*libm.a*), ya que estas funciones no son de uso general y no se han incluido por defecto en la estándar. Ocurre lo contrario con las librerías de manejo de la entrada/salida o las funciones de strings, que sí se incluyen por defecto al realizar los ejecutables (podemos considerar una librería como un conjunto de ficheros que contienen código objeto, las propiedades de esos ficheros son asimiladas en la propia librería que los mantiene a través de un registro índice).

Otra pregunta que surge es: ¿Puedo yo crear mis propias librerías? La respuesta es sí. Existe en sistemas Linux (UNIX) el comando *ar* para ello, que nos permite crearlas (añadir módulos), modificarlas o eliminarlas (quitar módulos).

Como cualquier comando, la sintaxis del mismo incluye opciones y argumentos:

```
ar -[opciones] [módulos] librería [ficheros]
```

[1] Gran parte de los inconvenientes de el código dependiente se pueden solventar con la compilación condicionada que nos proporciona el preprocesador. Referencias más amplias de él las podemos encontrar en el libro de Kernigham.

Las opciones más habituales son:

Opción	Significado
d	Borrar módulos a través de los ficheros indicados
m	Cambia de orden (mueve) un módulo en la librería
p	Pinta en pantalla un módulo a través de su fichero
q	Añade de forma rápida módulos (sin registro índice) al final
r	Reemplaza módulos a través de su fichero
t	Muestra el contenido de la librería
x	Extrae módulos a través de su fichero
o	Preserva la fecha original del módulo en la extracción
s	Crea o actualiza el registro índice
u	Reemplaza teniendo en cuenta la fecha
Modificador	
a	Lo coloca detrás de un módulo existente
b, i	Añade delante de un módulo existente
c	Crea una librería
v	Modo "verbose"

De esta manera si tenemos una librería que se llama *libre.a* y tres módulos *mod1.o*, *mod2.o* y *mod3.o* podemos hacer:

Ejemplo	Acción
ar c libre.a	Crea la librería
ar r libre.a mod1.o	Añade el módulo y crea la librería si no existe
ar tv libre.a	Muestra el contenido de la librería
ar q libre.a mod2.o mod3.o	Coloca al final de forma rápida el módulo
ar s libre.a	Actualiza el registro índice
ar x libre.a mod3.o	Extrae el tercer módulo

A la hora de usar la librería creada tenemos que tener en cuenta las siguientes reglas:

1. Las librerías se buscarán en los directorios por defecto que son "lib" y "/usr/lib". Si no ponemos nuestra librería ahí, tendremos que utilizar la opción del compilador -L para indicarlo.
2. Lo mismo tendremos que hacer con los ficheros incluidos de cabecera, en este caso la opción es -I.
3. Todas las librerías que creemos las empezaran con la palabra "lib" a la que seguirá el nombre propio de la librería (en el caso anterior "re").
4. Al compilar tendremos que invocar al enlazador con la opción -l para que incluya la librería creada (en el ejemplo -lre).

Existe un comando relacionado con las librerías, comando *nm*, para ver el contenido de sus módulos. La sintaxis es:

```
nm -[opciones] [ficheros]
```

donde el fichero puede ser un módulo o una librería, en este último caso se puede usar la opción -s para ver el índice.

El editor universal de UNIX / Linux: vi (vim)

En todos los sistemas UNIX y formando parte del mismo como un comando más, existe una familia de editores que nos permitirán procesar texto. Esta familia procede del editor ed, que era el editor original del UNIX, pero que prácticamente ya no se usa.

La familia consta de tres miembros, por un lado el ex que es la versión moderna y mejorada del ed, el edit que es un subconjunto del ex para principiantes, y el vi, que a diferencia de los otros editores que eran de línea, es de pantalla (un editor de línea sólo trabaja con una línea, por lo que es muy incómodo de usar, un editor de pantalla nos permite trabajar en todo el texto, aunque sólo nos muestra una ventana del mismo). Este último además realiza llamadas al ex para realizar determinadas funciones.

En muchos sistemas operativos existen otros editores, como el emacs o el mismo WordPerfect, ya que el vi es muy poco amigable (es desagradable de usar), pero tienen la desventaja que no son universales, cuestión que en el vi está garantizada ya que forma parte del sistema operativo.

Todos los editores funcionan de igual manera, es decir, a través de un buffer en memoria. Cuando se lee un fichero desde el editor, este se carga en memoria en un buffer sobre el cual se hacen las modificaciones y una vez terminado el trabajo se vuelca el contenido de la memoria en el fichero. Esto tiene una ventaja y es que si se hace algo mal podemos dejar el trabajo sin salvar (pasar a disco), pero también tiene una gran desventaja, y es que si ocurre una fallo en la corriente o en el sistema, todo el trabajo realizado sobre la memoria se perderá, por eso es muy importante realizar volcados a disco cada cierto tiempo (hay editores avanzados que lo hacen automáticamente como el WordPerfect).

El editor vi también trabaja con un buffer en memoria que conviene salvar cada cierto tiempo. Para ello tendremos que ponerlo en modo comando y hacer ":w", pero) qué significa modo comando ? El vi tiene dos modos de trabajo, uno donde introduciremos caracteres que formarán parte del texto y otro donde esos caracteres son acciones sobre el editor, como veremos en el siguiente apartado.

Funcionamiento

Lo primero que tenemos que hacer para trabajar con el editor es evidentemente ejecutarlo, para ejecutar el vi tendremos que poner su nombre seguido opcionalmente por el nombre de un fichero de texto que queramos editar:

```
$ vi fichero
```

si el fichero ya existe aparecerá en pantalla el mismo, si es nuevo aparecerá una tilde "~" en las líneas que estén sin ocupar, en este caso la primera, donde se realizará la escritura del texto.

Cuando ejecutemos el vi nos lo encontraremos en modo comando, con lo cual esperará que le introduzcamos una orden (carácter) sobre la que actuar, bien cargar un fichero si es que no se ha indicado ninguno en la línea de ejecución, o bien un comando de paso a modo texto.

Para pasar a modo comando de nuevo se utiliza un carácter especial que es el de escape y que se puede conseguir con la tecla de escape (no hay ningún problema en pulsar varias veces esa tecla, a parte de un pitido por cada par de pulsaciones). En la pantalla del vi sólo aparece el texto sobre el que se escribe, por lo que no sabemos (con algún indicativo en pantalla) en que modo estamos, por lo que se suele pulsar varias veces la tecla escape para garantizar el modo comando, esto unido a lo anterior, hace de una sesión de varios usuarios de vi, un concierto de pitidos.

Comandos

El vi tiene casi un centenar de comandos, de los que habitualmente sólo se utilizan una docena. Estos comandos se pueden dividir en varios grupos dependiendo de la función a realizar.

Movimiento del cursor

Existen más de 40 comandos de este tipo, pero los principales son cuatro, indicativos de las cuatro direcciones sobre las que nos podemos mover:

```
h : izquierda
l : derecha
j : abajo
k : arriba
```

la dirección a la que apuntan es la misma que la posición que ocupan en el teclado, restringiéndose su movimiento a la zona ocupada por el texto (si intentamos salir de esa zona nos avisará con un pitido). También se podrán utilizar las flechas del teclado.

Hay que incluir dentro de los comandos de movimiento la combinación de caracteres avance de línea (10) y retorno de carro (13), que se consiguen con la tecla return o enter. Esta servirá para dar a una línea por concluida y pasar a la siguiente en la primera posición.

Los otros comandos de movimiento menos usados aparecen en la tabla II, donde "^" significa que se debe pulsar antes del comando la tecla de control (un carácter de control es otro carácter más)

```
b  :    principio palabra (begin)
e  :    final de la palabra (end)
0  :    comienzo de línea
$  :    final de línea
^d :    12 líneas abajo (down)
^f :    24 líneas abajo (forward)
^u :    12 líneas arriba (up)
^b :    24 líneas arriba  (back)
nG :    ir a la línea n (go)
^g :    número de línea actual
```

Entradas a modo texto

Fundamentalmente hay tres comandos para escribir texto, el *a* para añadir a partir de la posición del cursor, el *i* para insertar delante del cursor y el *o*, *O* para añadir una nueva línea.

```
a :     añadir (append)
i :     insertar (insert)
o :     abrir debajo una línea
O :     abrir arriba una línea
```

Hay que recordar que en todo el entorno Unix las mayúsculas y las minúsculas son diferentes, por eso también existen las variantes A y I, la primera añadirá texto al final de la línea actual y la segunda insertará texto al comienzo de la misma. Si queremos escribir caracteres especiales antes tendremos que señalarlos con el carácter ^v.

Borrado y alteración

Una vez que hayamos escrito algo, ese texto se puede borrar o alterar pasando a modo comando. Para borrar se utilizan dos comandos fundamentales (y sus variantes) el *x* y el *d*. El x para borrar el carácter sobre el que estamos y el d (que se suele utilizar modificado) para borrar y pasar a una zona de memoria especial donde se puede recuperar lo borrado.

```
x :   borra un carácter
d :   borrar poniendo en el buffer
r :   reemplazar un carácter (replace)
R :   reemplazar varios caracteres
c :   borrar e insertar (change)
```

El *r* es el comando de hacer sustituciones de texto, si utilizamos la versión minúscula, sólo podremos sustituir un carácter, pasando de nuevo a modo comando, si utilizamos la mayúscula nos pondremos permanentemente en modo sustitución (hasta que con escape pasemos a modo comando), con lo que podremos sustituir varios caracteres. El comando *c* es una combinación del comando *d* y el *i*, primero borra lo indicado y después pasa a modo inserción para escribir de nuevo lo borrado.

Deshacer cambios

En caso de error, existe un comando para deshacer los cambios realizados, este es el *u* (undelete), del cual hay la versión minúscula para deshacer un sólo cambio o la mayúscula para deshacer todos los de la línea.

```
u : deshacer un cambio (undelete)
U : deshacer todos los cambios de la línea
```

Salida

Una vez que hayamos realizado la edición, normalmente querremos salvar nuestro trabajo en un fichero y abandonar la edición. Existen desde el vi varias formas, la más usual es la que parece a continuación:

```
ZZ : salva en el fichero y sale
```

Separación y unión de líneas

Hay dos comandos fundamentales para cortar y unir líneas, el primero y obvio es el retorno de carro, el segundo es J (join), observar que se utiliza la mayúscula.

<enter> : separar dos líneas
J : unir dos líneas (join)

Copiado y recuperación

A parte del fichero y del buffer temporal donde se realiza el trabajo de edición, hay una serie de buffers temporales auxiliares donde se colocan los últimos cambios o borrados que hayamos hecho, y hay una serie de comandos que nos permiten trabajar con estos buffers para hacer copias de texto.

y : copiar dejando en el buffer (yank)
p : poner el buffer en el texto después (put)
P : poner el buffer en el texto antes

Hay fundamentalmente tres comandos que nos permiten trabajar con estos buffers, por un lado está el comando *d* que ya hemos visto, y con el cual realizamos borrados de texto que colocamos temporalmente en un buffer. Y por otro lado están el comando *y* (yank) y el *p* (put), el *y* copia el texto seleccionado y lo coloca en un buffer, y el *p* hace lo contrario, coge la información del buffer y la coloca en el texto.

Modificadores de comandos

Hay ciertos comandos, sobre todo los del último apartado, que casi nunca se utilizan solos, sino que lo hacen con sus modificadores. Hay varios tipos:

- Repetir el comando: Esto hace que el comando actúe en toda la línea. Así si *d* borra, *dd* borra toda una línea.
- Poner un número delante del comando: Hace que ese comando se repita tantas veces como el número indicado. Así, *4dd* borrará 4 líneas.
- Alcances: Hay una serie de caracteres que después de un comando modifican el alcance del mismo. Estos alcances son:

e :	hasta el final de la palabra.
w :	hasta el comienzo de la siguiente palabra.
b :	hasta el comienzo de la palabra.
$:	hasta el final de la línea.
0 :	hasta el comienzo de la línea.
) :	hasta el comienzo de la siguiente frase.
) :	hasta el comienzo de la frase.
} :	hasta el final del párrafo.
{ :	hasta el comienzo del párrafo.

Así, *de* borrará hasta el final de la palabra donde esté situado el cursor, o *d$* borrará hasta el final de la línea.

Cualquiera de estos modificadores puede utilizarse conjuntamente con otros, de esta manera *2dw* borrará dos palabras.

Hemos dicho que con estos comandos se utilizan unos buffers especiales donde se colocan los cambios que estamos realizando, en vi no sólo existe un buffer para realizarlos, sino que hay varios que se nombran con los números y letras del alfabeto, aunque lo normal es utilizar el buffer por defecto.

Para utilizarlos se pone delante del comando unas comillas, el código del buffer, y por último el comando (incluso con otros modificadores).

Así, *"c4dd* borra cuatro líneas y las coloca en el buffer c. Para recuperar las cuatro líneas borradas bastaría hacer: *"cp*. Recordad que si no nos interesa lo recuperado, siempre podemos volver atrás con el comando *u*. Esta forma de utilizarlo puede servirnos para ir inspeccionando los buffers uno a uno.

Llamadas al editor de líneas ex

El vi está capacitado para utilizar los comandos del editor de líneas ex, de tal manera que tenemos otro conjunto de comandos a los cuales se accede en modo comando anteponiéndoles el carácter ":". De echo se puede pasar de un editor a otro utilizando el comando Q (pasa al ex) y el : (pasa al vi). En muchos casos estos comandos son tan importantes como los originales del vi.

Además se pueden utilizar varios comandos juntos como: *:r !who* que ejecuta el comando *who*, lo lee, y lo coloca en el fichero de edición.

Cabe destacar que los comandos del ex admiten delante de ellos dos números que son el rango de filas donde actúan, así, si escribimos *:3,5w fic*, estamos haciendo que se escriban las líneas 3, 4 y 5 en el fichero fic. Lo mismo se puede hacer con *:co* y *:m*, para *:r* sólo hace falta poner la línea detrás de la cual se colocará el fichero leído. Para indicar todas las líneas del texto se utiliza la *g*, que significa que se hará globalmente, esta opción es muy importante en los comandos que se verán posteriormente.

Los comandos de búsqueda y sustitución son muy utilizados. Estos se hacen en combinación con el ex. Los dos comandos del vi para realizar estas tareas son el */* y el *?*. El primero busca un patrón hacia adelante y el segundo hacia atrás. Además tenemos el comando *n* que repite la búsqueda. A continuación aparece una tabla de los más utilizados:

Comando	Descripción	Ejemplo
:wq	Salva el texto en un fichero y sale del editor	:wq
:w fichero	Escribe el texto en un fichero	:w pepe
:q!	Sale del editor sin salvar el texto	:q!
:r fichero	Lee un fichero y lo coloca en el cursor	:r pepe
:! comando	Ejecuta un comando de la shell desde el editor	:! who
:co	Copia unas determinadas filas	:3,5co7
:m	Mueve unas determinadas filas	:4,8m9

La estructura de un comando de este tipo es la siguiente:

```
:dirección/orden/parámetros
```

donde dirección consta de las líneas donde se debe realizar la acción y el patrón (secuencia de caracteres) a buscar, orden será lo que se quiere hacer después de encontrado el patrón, y parámetros normalmente el número de veces que se quiere realizar la operación. Veamos unos ejemplos:

:1,7/can /s//perro /	Se substituye de la 1 a la 7 can por perro una vez.
:g/can /s//perro /g	En todo el fichero y todas las veces.
:/patrón/s/patrón/nuevo/	Busca patrón y sustituye por nuevo
:g/p/s/p/n/	Busca p y lo sustituye por n globalmente

En el primer ejemplo la dirección serían las líneas 1 a 7 y el patrón *can* , una vez buscado se dice con la orden *s//* que se sustituya ese patrón (habría que repetirlo pero por defecto es el mismo, por lo tanto se escribe *//*) por el nuevo patrón *perro* . Si se coloca como parámetro la *g* se realizará la sustitución en todos los encuentros.

Por último hay que decir que desde el vi se pueden editar varios ficheros a la vez (uno detrás de otro), para ello se deberá ejecutar el vi con varios ficheros contiguos: vi f1 f2 f3. Los comandos son:

:e fichero	Editar otro fichero
:n	Pasar al siguiente fichero en edición múltiple

Opciones del editor

Por último y brevemente, hay que decir que el vi se puede personalizar y ponerle otras opciones como los espacios por tabulador, o el número de columnas de texto. Para ello se utiliza el comando del ex *:set*.

Las opciones más comunes son:

ai	identar.
ic	ignorar mayúsculas.
nu	numerar las líneas.
redraw	reescribir.
sw	número de espacios en ai.
ts	número de espacios en tab.
w	número de líneas de texto.
wm	margen derecho (80-x).
all	ver todas las opciones.

Un ejemplo de utilización de este comando, y quizás el más útil, sería para poner delante del texto como referencia en número de la línea:

```
:set nu
```