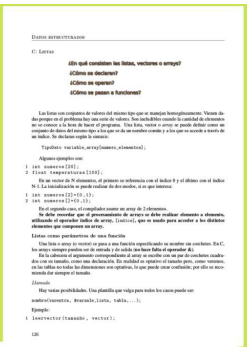
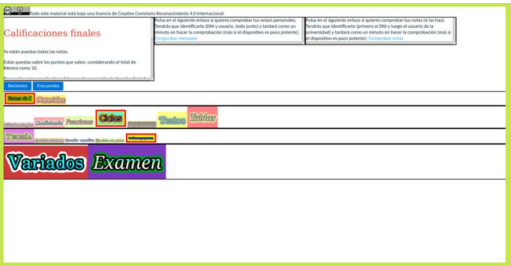
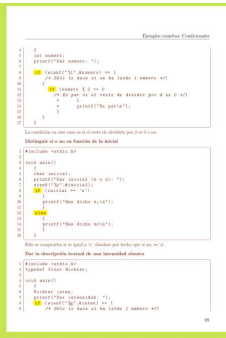
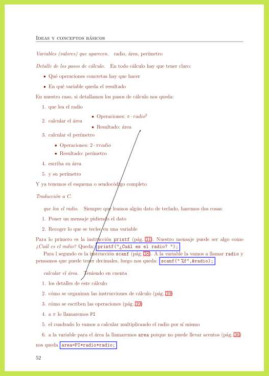
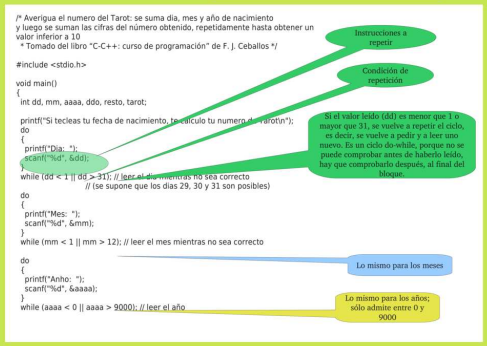


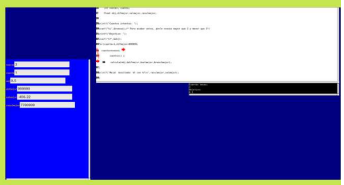


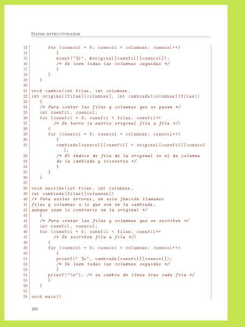


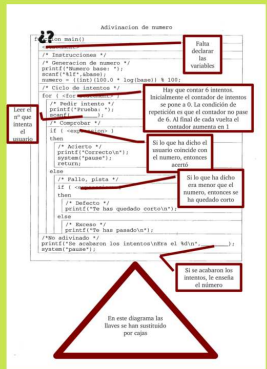


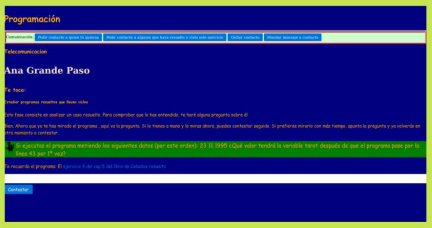


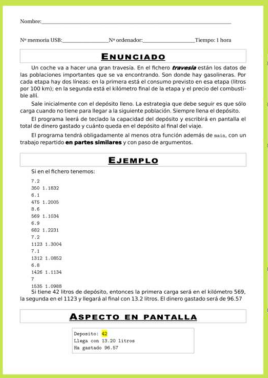
Fundamentos de 2024-2025 Computación











AGRADECIMIENTO

A Marta E. Zorrilla Pantaleón, que elaboró el primer volumen de apuntes y a Luis Crespo Ruiz, por revisar, corregir erratas y aportar sugerencias.

Índice general

Índice de figuras

Índice de tablas

Parte I

Ordenador y sistema operativo

Ordenador y sistema operativo

Niveles de trabajo

Los programas con los que trabajamos habitualmente están realizados en lenguajes preparados especialmente para que las personas puedan usarlos para programar. Se denominan lenguajes de alto nivel.

Estos lenguajes no son comprensibles para el procesador, que maneja un lenguaje más sencillo, llamado lenguaje de nivel máquina.

Para ejecutar un programa escrito en alto nivel hay que traducirlo a nivel máquina. Una parte de las instrucciones se traduce así, pero otras afectan o pueden afectar al funcionamiento general del ordenador y por ello pasan por una capa de control, que es el nivel de sistema operativo.

Las funciones o programas del sistema operativo deberán también ser traducidos a nivel máquina, porque es lo único que conoce el procesador.

Dentro del procesador, cada instrucción de nivel máquina se hace en varios pequeños pasos, denominados microinstrucciones, que implican a componentes internos del procesador.

En resumen, los programas están escritos en lenguajes de alto nivel, parte de cuyas instrucciones son intervenciones del sistema operativo; al final todas hay que pasarlas a lenguaje máquina, cada instrucción de éste consiste en una serie de microinstrucciones elementales y éstas se consiguen con circuitos de lógica digital, que se implementa con componentes electrónicos.

Por ejemplo, una acción de un programa requiere ejecutar 100 instrucciones de alto nivel. Cuatro de ellas son intervenciones o llamadas al sistema operativo. Si cada una de estas llamadas acaba traduciéndose en 100 instrucciones máquina y cada una de las otras instrucciones de alto nivel se acaba traduciendo en 10 instrucciones de nivel máquina, al final habrá que ejecutar $4 \times 100 + 96 \times 10 = 1360$ instrucciones de nivel máquina. Si cada una de éstas se resuelve en 5 microinstrucciones, el número total de microinstrucciones será de 6800.

El procesador va ejecutando las microinstrucciones al ritmo de un circuito reloj. Cada lapso de tiempo marcado por ese reloj se denomina ciclo. Así una instrucción máquina se resolverá en varios ciclos; a ese valor se le denomina CPI (Ciclos Por Instrucción). El reloj antedicho genera los ciclos a una cierta velocidad denominada frecuencia de trabajo, que se mide en Hertzios (ciclos por segundo) o, comúnmente, múltiplos.

En el ejemplo comentado arriba el CPI era de 5. Si la frecuencia es de 1 MHz, el total de microinstrucciones se resolverá en $\frac{6800}{1000000} = 0{,}0068\text{s} = 6{,}8$ milésimas de segundo.

Bibliografía (para este apartado): [11, cap. 1 y 7], [2, cap. 2 y 7] , [3, cap. 1 y 15] , [6, cap. 1] y [5, cap. 1]

Medida de tamaño de información

En el ordenador cualquier información se pasa a números y esos números los maneja internamente en el sistema de numeración binario, que es un sistema de numeración en que las cifras sólo pueden valer 0 ó 1 (dentro del ordenador serán tensiones). Vamos a ocuparnos de las unidades para medir cuánto ocupa una información almacenada en el ordenador.

Partimos del mínimo absoluto: una cifra binaria. Su nombre es bit, que es una contracción de las palabras inglesas "binary digit" (en nuestro idioma: cifra binaria).

Por razones históricas se maneja mucho un múltiplo que se llama byte. 1 byte = 8 bits

Una unidad especial es el tamaño que tienen las direcciones de memoria, que es el que el procesador opera de una vez. Varía de modelo a modelo, por lo que no tiene una equivalencia fija y nos lo tienen que decir para cada equipo. A esta unidad se le llama "palabra".

El resto de múltiplos son los habituales en el Sistema Internacional de Unidades, con la salvedad, que por razones operativas relacionadas con el binario, suelen ser convenientes múltiplos cuyo factor de conversión sea una potencia de 2. Sucede que 1024 es 2 elevado a 10 por lo que se han usado históricamente los prefijos Kilo, Mega, Giga, etc. trabajando con factor 1024 en vez de 1000. Para evitar confusiones está empezando a extenderse una nomenclatura diferenciadora, que es la que presentamos aquí. Llegamos hasta múltiplos suficientemente grandes, aunque la lista de prefijos teóricos es mayor.

Nombre	Abreviatura	Equivalencia
kilobyte	kB	1 kB = 1000 bytes
kibibyte	kiB	1 KiB = 1024 bytes
Megabyte	MB	1 MB = 1000 kB
Mebibyte	MiB	1 MiB = 1024 KiB
Gigabyte	GB	1 GB = 1000 MB
Gibibyte	GiB	1 GiB = 1024 MiB
Terabyte	TB	1 TB = 1000 GB
Tebibyte	TiB	1 TiB = 1024 GiB
Petabyte	PB	1 PB = 1000 TB
Pebibyte	PiB	1 PiB = 1024 TiB

Gestión de memoria

En este curso sólo nos ocuparemos de la asignación de memoria de una forma simplificada.

Asignación de memoria principal

Dentro de un programa las instrucciones se refieren a otras instrucciones o a variables. Para ello usan su posición en la memoria. Estas posiciones están numeradas de 0 en adelante y se llaman "direcciones".

Típicamente, para optimizar el uso de la memoria, ésta está dividida en bloques de un tamaño fijo, llamados "páginas". A cada proceso se le divide en páginas, que se van colocando en memoria según convenga. Para evitar la confusión entre las páginas del proceso y las de

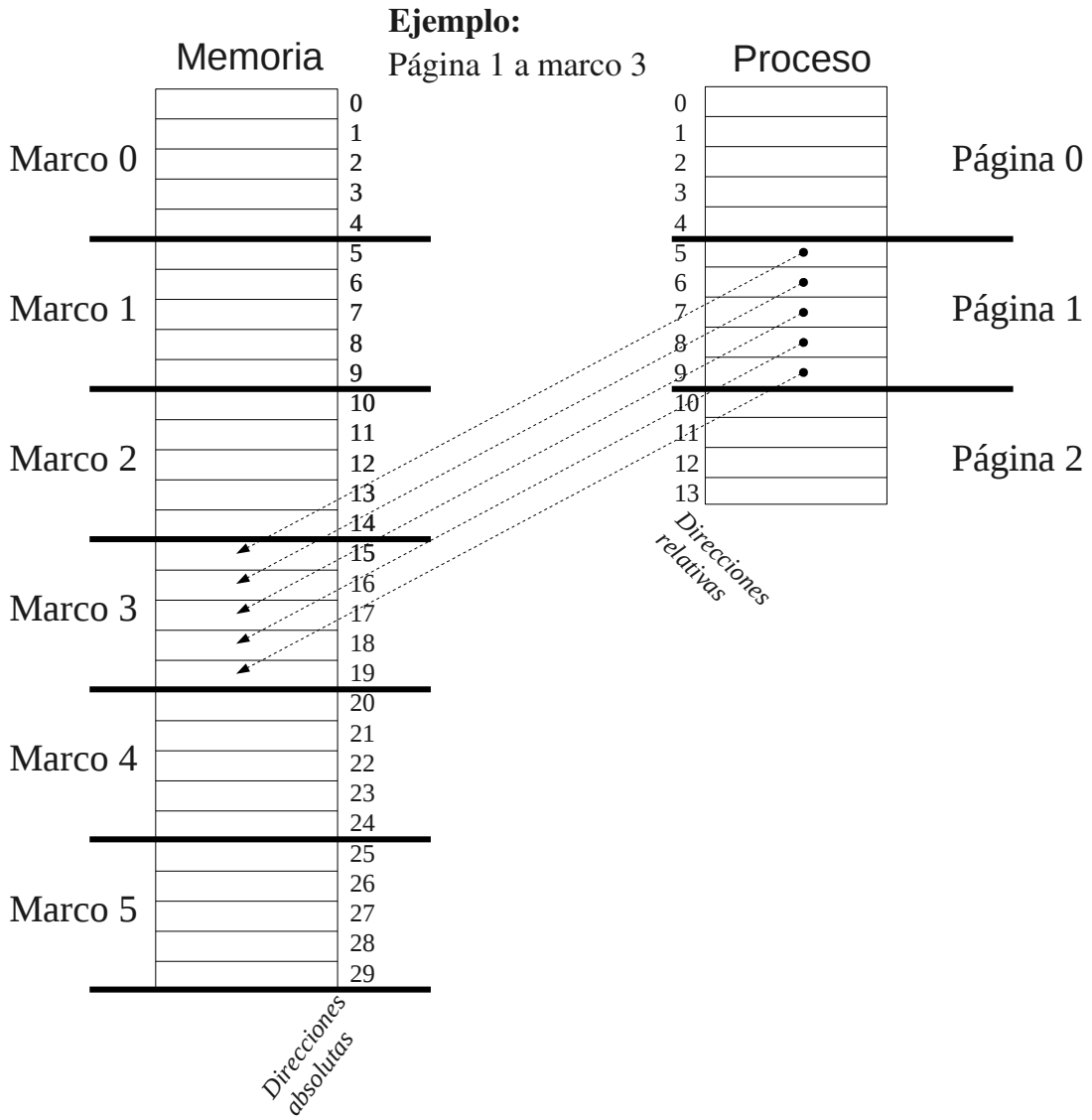


Figura 0.1: Ejemplo con tamaño de página de 5 palabras

la memoria, a éstas últimas las denominaremos "marcos". Según los procesos se van ejecutando, aparecen y desaparecen marcos libres en la memoria principal, pero no necesariamente contiguos, por lo que el proceso no recibe una asignación contigua en la memoria.

Un ejemplo con números pequeños se ilustra en la figura 0.1.

Veamos un caso: Si en cada página (o marco, ya que son del mismo tamaño) hay, por ejemplo, 100 direcciones, la dirección relativa 130 del proceso, estará en la segunda página, ya que la primera página irá de la 0 a la 99 y la segunda de la 100 a la 199. Ahora, si esa página ha ido a parar al cuarto marco de la memoria, entonces las direcciones absolutas en ese marco irán de la 300 a la 399. Por lo tanto la dirección relativa 130, que está 30 más adelante que el inicio de página, habrá ido a parar a la dirección absoluta 330.

Gestión de tareas

Multiproceso

En un sistema multiproceso puede haber varios procesos trabajando simultáneamente. El problema que debe resolver el sistema operativo es el reparto del tiempo del procesador.

La primera fase de la planificación, llamada de medio plazo, es cargar en la memoria los procesos que están listos para su ejecución. La segunda, de corto plazo, es decidir cuál de ellos pasa a ejecutarse, y durante cuánto tiempo.

Si a los procesos se les asigna todo el tiempo que piden puede que un proceso largo tenga colgados a todos los demás demasiado tiempo, o que un proceso que quede mal según el criterio utilizado nunca llegue a ejecutarse, porque van llegando otros que se ponen por delante. Para evitar esto se utiliza una ejecución por intervalos: a cada proceso se le asigna, no todo el tiempo que pida, sino hasta un intervalo máximo que llamaremos "ventana de ejecución". Para ello se utiliza un contador de tiempo que tiene el procesador y que al cubrirse el tiempo, da paso al sistema operativo, parando el proceso en curso (mediante lo que se denomina una "interrupción"). Así, cada vez que se consume un intervalo vuelve a tomar el control el planificador del sistema, que decide qué proceso va a ejecutarse después. Puede hacerse simplemente por turno a todos los procesos, o, si hay distintas prioridades, por turno a los de alta prioridad hasta que no haya ninguno activo y luego por turno a los de baja. O también combinarse con los esquemas anteriores, ordenando la cola de procesos no por turno sino por otro criterio.

Por ejemplo, si tenemos una ventana de 3 ms y están en la cola de ejecución los procesos A, B y C y despreciamos el tiempo que tarda el sistema en cambiar de uno a otro, tendríamos que el proceso A está 3 ms, los siguientes 3 son para el B, los siguientes 3 para el C, los siguientes 3 otra vez para el A, otros 3 para el B, etc.

Ejercicios

Haz los siguientes ejercicios. Comprueba las soluciones propuestas. Si no consigues casarlo, consulta con los profesores.

1. Se tienen los siguientes procesos, con los tamaños que se indican, medidos en instrucciones:

Winword 8605
soffice 212
Iexplore 89
k-meleon 564

Las páginas son de 40 instrucciones . La memoria principal tiene un tamaño total de 200 instrucciones.

Se van registrando las siguientes peticiones (proceso y número de página que pide, numeradas desde 0):

Winword 0, Winword 100, soffice 0, soffice 5, Winword 30, Iexplore 0, Iexplore 2, soffice 4, Iexplore 1, k-meleon 0, Winword 5

Decir las direcciones absolutas y relativas de la primera instrucción de cada proceso que está cargada en la memoria principal en cada momento, si cada proceso sólo puede tener una página en memoria principal y se va asignando desde el principio por orden de petición.

Respuesta (absoluta,relativa):

Winword	0,0	0,4000			0,1200						0,200
soffice			40,0	40,200				40,160			
Iexplore						80,0	80,80		80,40		
k-meleon										120,0	

2. En un sistema se tienen los siguientes procesos, prioridades de salida, y tiempo que trabajan

Winword baja: 30

soffice baja: 20

Iexplore alta: 20

k-meleon alta: 25

El planificador maneja una cola de baja prioridad, con intervalo de tiempo de 5, alternada con otra de alta, con intervalo de 10; el tiempo que tarda el planificador en decidir el siguiente es 2 (cada vez que se cambia de proceso, el planificador entra y le lleva ese tiempo organizar al siguiente); el criterio que se utiliza para elegir es el de proceso que lleve más tiempo parado. Indicar qué va haciendo el procesador en cada momento.

Respuesta:

Tiempo	Proceso
0	planificador
2	Iexplore
12	planificador
14	Winword
19	Planificador
21	K-meleon
31	Planificador
33	Soffice
38	Planificador
40	Iexplore
50	Planificador
52	Winword
57	Planificador
59	K-meleon
69	Planificador
71	Soffice
76	Planificador
78	K-meleon
83	Planificador
85	Winword
90	Planificador
92	Soffice
97	Planificador
99	Winword
104	Planificador
106	Soffice
111	Planificador
113	Winword
118	Planificador
120	Winword

3. Un programa tiene que ejecutar trescientas mil instrucciones de alto nivel. Se sabe que cada instrucción de alto nivel que pasa a directamente nivel máquina equivale en promedio a 11 instrucciones de nivel máquina, y que cada llamada al sistema operativo se resuelve en promedio con 500 instrucciones de nivel máquina. El procesador tiene un CPI de 7,5 y una frecuencia de 400 Mhz. El programa se ejecuta en 8 centésimas de segundo. ¿Cuántas de las instrucciones de alto nivel pasaban por el sistema operativo?

   Respuesta: 1977

Bibliografía

[1] P. B. Galvin A. Silberschatz. Fundamentos de sistemas operativos. McGraw-Hill, 2006.

[2] V. Feliu C. Cerrada. Estructura y tecnología de computadores I. Universidad Nacional de Educación a Distancia, 1995. Hay algunos ejercicios propuestos, así como ejemplos. Existe una parte importante del libro que no es aprovechable en este curso, por desarrollar en detalle un lenguaje ensamblador. Hay una edición del 2001 con el título "Fundamentos de estructura y tecnología de computadores".

[3] P. de Miguel. Fundamentos de los computadores. Paraninfo, Madrid, 1996. Libro completo sobre funcionamiento del ordenador, tanto teórica como prácticamente. De nivel muy superior a la asignatura, con ejemplos y bastantes ejercicios. Útil para ver con más profundidad y desarrollo los conceptos presentados. Hay una edición del 2004.

[4] D. M. Dhamdhere. Sistemas operativos. Un enfoque basado en conceptos. McGraw-Hill, 2008.

[5] A. Prieto et al. Introducción a la informática. McGraw-Hill, 2001. Libro también profundo y completo sobre informática. Con bibliografía y ejercicios propuestos tras cada tema. Con explicaciones amplias. Hay una edición del 2006 y un libro algo más reducido del 2005 bajo el título "Conceptos de informática".

[6] J. Mª Angulo P. de Miguel. Arquitectura de computadores : fundamentos e introducción al paralelismo. Paraninfo, 1995. Libro muy completo sobre arquitectura de ordenadores. También de nivel muy superior a la asignatura, con ejemplos y ejercicios; posiblemente algo menos teórico que el anterior. Para los alumnos tiene una utilidad similar.

[7] W. Stallings. Organización y arquitectura de computadores : diseño para optimizar prestaciones. Prentice Hall, 2000. Libro bastante completo sobre arquitectura de ordenadores. Con bibliografía y ejercicios propuestos tras cada tema. Tiene bastante más información que la requerida por los alumnos; puede considerarse que la compensación entre legibilidad y profundidad es buena. Hay una edición del 2006.

[8] W. Stallings. Sistemas operativos: aspectos internos y principios de diseño. Prentice Hall, 2005.

[9] A. S. Tanenbaum. Organización de computadores. Un enfoque estructurado. McGraw-Hill, 1992. Esta enfocado a los niveles de orgamización y la parte de sistema operativo es uno de los niveles jerárquicos de descripción (capítulo 6). Hay una edición del 2000.

[10] A. S. Tanenbaum. Sistemas operativos modernos. Pearson, 2009.

[11] E. Alcalde y M. García. Informática básica. McGraw-Hill, 1995. Libro de presentación de la informática en general; puede ser útil para una lectura básica que centre las ideas generales.

[12] J. Aranda y otros. Sistemas operativos. Sanz y Torres, 2002.

[13] J. Carretero y otros. Sistemas operativos: una visión aplicada. McGraw-Hill, 2007.

[14] V. C. Hamacher y otros. Organización de computadores. McGraw-Hill, 1987. Descripción muy breve y resumida del sistema operativo en la sección 10.4. Hay una edición del 2003.

Parte II

Bases de datos

Introducción a bases de datos

Conceptos básicos de bases de datos

Una base de datos es una colección homogénea de informaciones referidas a una serie de entes.

Ejemplos

- Datos de compras de una fábrica: remesas, proveedores
- Hotel: clientes, habitaciones, tarifas
- Piezas: composición de un aparato o mueble, relaciones entre ellas
- Hospital: pacientes, camas ...
- Universidad: alumnos, asignaturas, ...

Tipos de bases de datos

Hay varios tipos de bases de datos. En este curso vamos a estudiar sólo los sistemas relacionales. La estructura de estos sistemas es que se tienen una serie de datos de una serie de entes (personas, productos, etc.). Están organizados en una serie de tablas (se las denomina relaciones) cada una con una serie de filas (que se llaman n-tuplas) para cada ente; en cada columna está un dato.

Ejemplo de diseño

Departamento de compras
Tablas:

1. Suministradores
    a) código
    b) nombre
    c) prioridad
    d) ciudad
2. piezas
    a) código

b) nombre

c) color

d) peso

e) almacén

3. remesas

a) código de suministrador

b) código de pieza

c) cantidad

d) coste

Uso

Las bases de datos están controladas por los programas específicos que se llaman programas gestores. Para decirle al programa gestor lo que tiene que hacer hay que utilizar un lenguaje; nosotros vamos a ver un lenguaje muy común llamado SQL.

SQL

En la bibliografía el SQL se presenta agrupado, incluyendo la parte de descripción y la de manipulación, por lo tanto, estas referencias son comunes: [6, cap. 7], [3, cap. 4] y [2, cap. 4].

Los textos en SQL se marcan con apóstrofes simples (’). Todas las instrucciones de SQL acaban en ;

Creación

El comando para crear nuevas tablas es `create table` cuya forma general es:

CREATE TABLE tabla(atributo tipo, atributo tipo, ...);

tabla es el nombre de la tabla; atributo es el nombre de cada atributo y tipo es la clase de valor que va a ir en ese atributo: puede ser TEXT, INTEGER, REAL,... Si se añade UNIQUE es que no se repetirá.

Ejemplo:

CREATE TABLE SUMINISTRADORES(CODIGO CHARACTER(5) UNIQUE, NOMBRE CHARACTER(50), PRIORIDAD INTEGER, CIUDAD CHARACTER(50));

Significado:

> Crear una tabla llamada SUMINISTRADORES donde va a haber un atributo CODIGO de cinco caracteres que no se repetirá, otro llamado NOMBRE de 50 caracteres, un entero llamado PRIORIDAD y otro texto de 50 caracteres máximo llamado CIUDAD.

Ejercicios

1. Crear una base de datos de biblioteca.

   Tabla de libros: ISBN, título, autor, editor, disponibilidad

   Tabla de usuarios: nombre, DNI, dirección

   Tabla de préstamos: clave, fecha, fecha de devolución

2. Preparar una tabla para clientes donde se almacenará el nombre, dirección completa, teléfono, FAX, descuento y zona. Otra para artículos con la descripción y el precio. Otra de pedidos donde además de identificar el cliente y el artículo se pondráa la cantidad y la fecha. Finalmente, una para describir las zonas.

Consultas

Hay distintas posibilidades que ofrece el SQL, pero sólo estudiaremos la instrucción SELECT.

El esquema básico de la instrucción es:

SELECT columnas FROM tablas WHERE condicion;

En columnas van los nombres de los campos que interesan separados por comas o un asterisco si se quieren todos. En tablas van los nombres de las tablas que interesan separados por comas. Cuando se usan varias tablas y hay columnas con el mismo nombre se puede indicar su nombre con el nombre de la tabla un punto y el nombre del atributo. En condición va el criterio de selección; se pueden utilizar los operadores habituales de menor y mayor, los de igual, distinto, el operador BETWEEN para seleccionar un intervalo y los operadores convencionales de combinación AND y OR. También se puede utilizar el operador LIKE para seleccionar por parecido; el patrón de parecido puede llevar el carácter porcentaje para indicar una secuencia arbitraria de caracteres y el guión de subrayado para indicar un carácter arbitrario.

Ejemplo complicado:

SELECT SUMINISTRADORES.CODIGO,PIEZAS.CODIGO FROM SUMINISTRADORES,PIEZAS WHERE SUMINISTRADORES.CIUDAD LIKE 'SAN%' AND SUMINISTRADORES.DESCUENTO+PIEZAS.EXTRA BETWEEN 3 AND 7;

Significado:

> Seleccionar los atributos CODIGO de la tabla SUMINISTRADORES y CODIGO de la tabla PIEZAS buscando en las tablas SUMINISTRADORES y PIEZAS las filas donde el campo CIUDAD de la tabla SUMINISTRADORES empieza por SAN y la suma de los campos DESCUENTO de SUMINISTRADORES y EXTRA de la tabla PIEZAS vale entre tres y siete.

Otras posibilidades interesantes son:

- Sacar, no los valores de los campos, sino su cuenta o su suma con los operadores COUNT o SUM o AVG
- Sacar sólo los que son diferentes con el operador DISTINCT

- Ordenar los resultados según algún criterio con el operador ORDER
- Juntar dos SELECT con el operador UNION

Ejemplo:

SELECT COUNT(CODIGO),SUM(LONGITUD+EXTRA) FROM PIEZAS WHERE EXTRA != 0;

Significado:

Mirar en la tabla PIEZAS y contar los códigos que tengan un Valor de EXTRA distinto de cero y la suma de todos los valores de LONGITUD y EXTRA de esas piezas.

Ejemplo:

SELECT DISTINCT * FROM PIEZAS ORDER BY LONGITUD DESC;

Significado:

Obtener todos los campos de todos los registros de la tabla piezas que sean distintos, ordenados descendentemente por longitud

Ejercicios

1. Sacar todas las filas y todas las columnas de la tabla Productos
2. Todas las filas y sólo las columnas Nombre, Numero y Precio de la tabla Productos
3. Las filas de Productos que tienen una línea de productos (Linea) de R y cuyo valor correspondiente a los días para fabricar (DiasFabricar) es inferior a 4
4. Los campos Puesto de Empleados evitando duplicados.
5. Se desea obtener los valores de DNI y el nombre de todos los de la tabla personas
6. Se desea obtener toda la información de quien tenga valor de DNI igual a 12453535Y en la tabla Personas
7. Se necesita DNI y nombre de quienes tengan primer-apellido BETANCUR y sexo M en la tabla personas
8. Extraer los diferentes valores de salario que haya en la tabla personas
9. Mostrar el total de registros en la tabla personas
10. Mostrar la cantidad total de quienes tienen codigo-dep con el valor 3 en la tabla proyectos
11. En la tabla proyectos hay datos de proyectos: entre otros, un campo codigo-dep que indica el departamento que lleva ese proyecto y otro numero-proy que es el número identificador del proyecto. En la tabla personas hay datos de empleados, entre otros un campo cod-dep para indicar el departamento en que trabaja esa persona. Se desean todos los datos que haya en la tabla personas de aquellos que trabajen en los departamentos que lleven los proyectos de nº 139001 y 139002

12. En la tabla personas hay datos de empleados, entre otros un campo salario para indicar el sueldo. Se quiere obtener el total pagado por la compañía, el máximo y el mínimo salario y el promedio

13. En la tabla proyectos hay datos de proyectos: entre otros, un campo nombre con el título y otro numero-proy que es el número identificador del proyecto. Se quiere el nombre y nº de proyecto de los que se titulen ...lic...

14. En la tabla personas hay datos de empleados, entre otros un campo salario para indicar el sueldo. Mostrar todos los sueldos de los que ganen menos de 1200000 aumentados un 18

15. En la tabla personas hay datos de empleados, entre otros nombre, primer-apellido y segundo-apellido. Listar todos los datos, ordenados por nombres y apellidos

16. En la tabla viajes-realizados hay datos de vuelos; entre otros, están los campos num-viaje, num-trayecto, fecha, cod-terminal-sale y cod-terminal-llega. Prepare una lista con los números de viaje y las fechas de todos los vuelos que salen del terminal codigo 'CA001' y llegan al terminal código 'BO001.

17. En la tabla suministradores tenemos los datos codigo, nombre, status (un número de prioridad) y ciudad. En la tabla piezas tenemos los campos codigo, nombre, color, peso y ciudad (donde está el almacén). En la tabla remesas tenemos codigo-suministrador, codigo-pieza, cantidad. Se desea conocer el viaje que han hecho (ciudad de origen y de destino) todas las remesas.

Estos ejercicios están basados en la descripción de tablas siguiente:

clientes

Nombre	Tipo
Numero	INTEGER
Empresa	TEXT
Direccion	TEXT
Ciudad	TEXT
Estado/provincia	TEXT
Codigo postal	CHAR(5)
Pais	TEXT
Telefono	INTEGER
FAX	INTEGER
TasaImpuestos	REAL
Contacto	TEXT
FechaUltimaFactura	TEXT

empleados

Nombre	Tipo
Numero	INTEGER
Apellido	TEXT
Nombre	TEXT
ExtensionTfno	INTEGER
FechaContrato	TEXT
Sueldo	REAL

piezas

Nombre	Tipo
Numero	INTEGER
NumeroProveedor	INTEGER
Descripcion	TEXT
Disponibles	REAL
Pedidos	REAL
Coste	REAL
Precio	REAL

pedidos

Nombre	Tipo
NumeroPedido	INTEGER
NumeroCliente	INTEGER
FechaVenta	TEXT
FechaEnvio	TEXT
NumeroEmpleado	INTEGER
NombreEnvio	TEXT
DireccionEnvio	TEXT
CiudadEnvio	TEXT
Estado/Provincia	TEXT
CodPostEnvio	INTEGER
PaisEnvio	TEXT
TfnoEnvio	INTEGER
EmpresaEnvio	TEXT
MomentoPago	TEXT
MedioPago	TEXT
Cantidad	REAL
Impuesto	REAL
TotalPagar	REAL

En cada uno se da como ejemplo lo que saldría usando la base de datos disponible desde la hoja de ejercicios del aula virtual. Puedes probarlo si quieres con `sqlite3`

1. Obtener la(s) persona(s) de contacto y teléfono(s) de los clientes de la ciudad de Granada.

   Ejemplo: Nos debe de salir

Persona	Teléfono
José Bal	915697044
Belén Tepes	809555689
Ana Ramos	809409002

Pistas:

Posible estrategia general En el SELECT hay que buscar persona y teléfono y en el WHERE, según elija el usuario se pedirá una ciudad

Teoría a repasar SQL

2. Obtener los nombres y teléfonos de los empleados que ganan entre 20000 y 24000.

   Ejemplo: Nos deben salir

Nombre	Teléfono
Pedro	888888888
Juana	222222222

3. Obtener los datos del pedido cuyo código es 1355.

   Ejemplo: Nos debe salir:

FechaVenta	1995-02-05
FechaEnvio	1995-02-05
NumeroEmpleado	141
EmpresaEnvio	UPS
MomentoPago	Instant
MedioPago	Tarjeta
Cantidad	13908
Impuesto	0
TotalPagar	13908

   Pistas:

   Posible estrategia general En el SELECT hay que seleccionar todos los atributos

4. Obtener la cantidad disponible y precio de todas las piezas cuya descripción termine en "Tank".

   Ejemplo: Nos deberá salir:

Numero	Disponible	Precio
9312	Sí	179
9318	Sí	195
9354	Sí	235
9316	Sí	325

   Pistas: Utilizar LIKE en el WHERE

Los siguientes ejercicios usan la tabla creada en el ejercicio 2 del apartado II

1. Listar artículos cuyo código esté en un intervalo (elige el que te parezca bien), ordenándolos por precio.
2. Listar clientes dado un intervalo de descuento (elige el que te parezca bien). Considera también los casos de sin límite superior o inferior.
3. Dada una zona por parte de su descripción (elige lo que te parezca bien), listar los clientes correspondientes a esa zona que hayan hecho algún pedido.
4. Listar los datos de pedido, incluyendo descripción del artículo, de aquellos clientes que sean de cierta ciudad (elígela)

Inserciones

Para la instrucción INSERT (añadir) el caso general es:

INSERT INTO tabla VALUES(valores,...);

tabla indica dónde queremos meter ese registro y los valores son los datos.
Ejemplo:

INSERT INTO PIEZAS VALUES('P5','Manivela','Rosa',0.014,'Jaen');

Significado:

Añadir en la tabla piezas un nuevo registro. Los valores van por el orden especificado para los atributos en la creación de la tabla.

Eliminaciones

Para la instrucción DELETE (borrar) el caso general es:

DELETE FROM tabla WHERE condicion;

Es como SELECT pero borrando los registros en vez de sacarlos.
Ejemplo:

DELETE FROM SUMINISTRADORES WHERE CIUDAD='MADRID';

Significado:

Borrar de la tabla SUMINISTRADORES todos los registros donde el atributo CIUDAD valga Madrid.

Modificaciones

Para la instrucción UPDATE (modificar) el caso general es:

UPDATE tabla SET atributo=valor, atributo=valor ... WHERE condición;

tabla indica la tabla en que queremos modificar. condición es lo que nos permite seleccionar los registros y mediante SET vamos indicando para cada atributo cuál es el nuevo valor que debe de tener.
Ejemplo:

```
UPDATE PIEZAS SET COLOR='Amarillo', PESO=PESO+0.1, ALMACEN=NULL
WHERE CODIGO='P2';
```

Significado:

> Cambia en la tabla piezas todos los registros donde el código sea P2, poniendo los valores de color a amarillo a peso sumarle 0.1 y borrar el campo de almacén.

Bibliografía

[1] E. Marcos A. de Miguel, M. Piattini. Diseño de bases de datos relacionales. Ra-Ma, 1999.

[2] S. Sudarshan A. Silberschatz, H. F. Korth. Fundamentos de diseño de bases de datos. McGraw-Hill, 2006. Relativamente denso, con una carga teórica no tan fuerte. Tiene resumen y ejercicios propuestos por capítulo.

[3] C. J. Date. Introducción a los sistemas de bases de datos. Pearson Educación, 2001. Muy completo y denso. Con resumen y ejercicios propuestos en cada capítulo, algunos resueltos; con una descripción inicial de cada capítulo. Entusiasta defensor del modelo relacional.

[4] A. de Miguel y otros. Diseño de bases de datos : problemas resueltos. Ra-Ma, 2000. En cada capítulo hay un esquema de la teoría y luego casos prácticos. Muy práctico y útil. Incluye los temas más concretos y habituales, centrado en su título; no incluye temas de ampliación.

[5] J. V. Hansen G. W. Hansen. Diseño y administración de bases de datos. Prentice Hall, 1998.

[6] S. B. Navathe R. Elmasri. Fundamentos de sistemas de bases de datos. Pearson Educación, 2008. Libro bastante completo y actualizado, con preguntas y ejercicios propuestos en cada capítulo.

Parte III

Programación

Guía de búsqueda rápida

Aquí tienes por nivel de programa y por tipo de material, las páginas de este volumen en que está cada cosa, o la primera de las páginas en que aparece, si aparece en varias. Si estás viendo el PDF en pantalla, las referencias deberían ser activas (pulsando en ellas te lleva)

Se ha diferenciado como casos explicados aquellos en que sobre el programa se han hecho algunas anotaciones, generalmente gráficas, para explicar los elementos de la solución, o se incluye una deducción más detallada de cómo se llega al programa.

	Teoría	Ejemplos de instrucciones aisladas	Preguntas de repaso	Plantillas
Cálculos directos	(pág. 39)	(pág. 45)	(pág. 46)	(pág. 45)
Ficheros	(pág. 59)			(pág. 59)
Condicionales	(pág. 64)	(pág. 68)	(pág. 69)	(pág. 64)
Ciclos simples	(pág. 78)	(pág. 83)	(pág. 83)	(pág. 104)
Ciclos cualesquiera				
Listas numéricas	(pág. 136)	(pág. 148)	(pág. 148)	(pág. 139)
Textos	(pág. 170)	(pág. 174)	(pág. 175)	
Tablas	(pág. 201)	(pág. 207)	(pág. 207)	(pág. 202)

	Casos explicados	Casos resueltos
Cálculos directos	(pág. 55)	(pág. 52)
Ficheros	(pág. 62)	(pág. 60)
Condicionales	(pág. 104)	(pág. 73)
Ciclos simples	(pág. 106)	(pág. 125)
Ciclos cualesquiera	(pág. 131)	(pág. 129)
Listas numéricas	(pág. 168)	(pág. 154)
Textos	(pág. 182)	(pág. 180)
Tablas	(pág. 227)	(pág. 214)

Ideas y conceptos básicos

En este tema vamos a repasar las ideas básicas de programación. Empezaremos viendo los conceptos para que nos suenen y comentándolos con un ejemplo.

Conceptos

Un programa es un conjunto de "instrucciones" ordenadas (Nunca pienses que porque algo esté explicado más abajo, él ya lo verá. No: si necesita una explicación en un punto, entonces tiene que verla antes) que se van ejecutando sucesivamente, junto con otra información que sirve para que el ordenador se aclare con las instrucciones.

Cada instrucción le indica al programa algo que debe hacer. Existen también instrucciones informativas, que no implican hacer nada, sino que sirven para que el sistema se aclare. En caso de duda, a las instrucciones que implican hacer algo se las llama ejecutables.

Este conjunto de instrucciones va a resolver el problema que tenemos. La resolución de cualquier problema partirá de unos datos y tiene que llegar a unos resultados. Tanto los datos como los resultados son unos valores de algún tipo que el programa tiene que manejar. Estos valores datos, resultados o intermedios, en el programa se llaman "variables".

Variables

Las variables son valores que el ordenador almacena. Tendrán un nombre, que será el que se usará en el programa para referirse a ellas. También hay que decir qué clase de valores podemos meter ahí: ¿números?, ¿letras?, ... Incluso puede tratarse no de un solo valor sino de varios, por ejemplo una lista de números, una palabra (que son varias letras), etc.

En cada momento (o sea, al llegar a cada instrucción) hay una serie de variables que tienen unos valores que es con lo que esa instrucción puede trabajar.

C: Estructura de programa

¿Qué pinta tiene el programa?

¿Qué elementos nos vamos a encontrar siempre?

¿Cómo se dice que escriba algo en pantalla?

La estructura de un programa en C consta de una o más funciones (bloques de instrucciones), no teniendo por qué estar todas en un mismo fichero. Una de estas funciones, `main()` es de obligatoria declaración en todos los programas C, ya que será la función por donde comience la ejecución del programa.

A continuación se presenta un programa sencillo para comentar cada una de sus partes. Como aclaración para casos de duda, como en la mayoría de listados del libro, las instrucciones demasiado largas se han partido en líneas para poder verlas; esto es para presentarlas en el papel, pero **no** se hará en la programación: las instrucciones no se pueden partir arbitrariamente en varias líneas[1]

```
1  #include <stdio.h>
2  #include <locale.h>
3
4  #define MENSAJE "Hola y adiós\n"
5
6  /* Este programa escribe un saludo por pantalla */
7
8  void main () {
9  setlocale(LC_ALL,"");
10     printf (MENSAJE);
11 }
```

En primer lugar se encuentra una directiva del preprocesador (`#include`) que incluye las definiciones de las funciones relativas a la entrada y salida (lectura y escritura) estándar `stdio.h`. Seguida hay otra análoga para cambiarle el idioma, ya que por defecto usa el inglés (`locale.h`)

A continuación se encuentra la zona donde se declaran unidades y las constantes, como MENSAJE, del programa.

[1] sólo pueden partirse en general después de las comas que separan variables o argumentos, o utilizando un carácter especial para fin de línea

- Los símbolos // y /* */ permiten introducir comentarios al código, facilitando así su
- compresión.

- A continuación se encuentra la función `main()`. Esta incluye todos los pasos que va a
- realizar el programa en su ejecución. En el ejemplo sólo incluye dos instrucciones:

- La primera sirve para que se olvide del inglés y coja el idioma del sistema (se supone
- que tenemos el sistema en nuestro idioma)

- La segunda, `printf`, que muestra un mensaje en la salida estándar, es decir, en pantalla,
- usando la constante MENSAJE. Aquí aparece \n que en C significa salto de línea.

C: Elementos básicos del lenguaje

Include

La directiva `#include` incluye un archivo en el archivo fuente actual. Normalmente se usa para incorporar ciertas funciones. La sintaxis usual de esta directiva es:

```
#include <nombre_archivo>
```

En cada capítulo se enumeran y describen las bibliotecas de funciones estándar C correspondientes que podemos usar en los include.

Comentarios

Los comentarios sirven para hacer más legibles los programas, no generan código ejecutable, es decir, que el compilador (el programa que traduce de alto nivel a nivel máquina) los ignora. Los comentarios comienzan con /* y finalizan con */ cuando ocupan más de una línea. También se puede utilizar // para comentar una sola línea.

Ejemplo:

/* Este es un comentario que
ocupa más de una línea */
// Este comentario ocupa una sola línea

C: Salida formateada

Función printf

Esta función escribe a pantalla.

Esquema:

printf(formato,arg1,. . . , argN);

Ejemplo:

```
printf("Hola y adios\n");
```

Vamos a empezar hablando de usarla sólo para texto.

Idioma y tablas de caracteres

Si pruebas esta escritura, sin más preambulos, verás que funciona perfectamente mientras uses caracteres del alfabeto inglés, pero si usas alguno que no lo sea (vocales acentuadas, eñes) sale mal. Vamos a ver cómo decirle que trabaje en el idioma local.

Lo primero es que tengas tu sistema en el idioma que quieres. Lo que viene a continuación va a referirse al idioma del sistema.

Arriba de todo añade: `#include <locale.h>` Ahí es donde tiene las definiciones de idiomas.

Luego, dentro de main, pon como primera instrucción: `setlocale ( ,"");` Con eso le dices que para todas (`LC_ALL`) las cuestiones que dependan del idioma vas a utilizar el del sistema (las comillas vacías).

Al decir para todas las cuestiones, esto incluye el separador decimal de los números. Si sólo queremos poder escribir mensajes con nuestro alfabeto pondremos `LC_CTYPE`, en vez de `LC_ALL`

Utilidades: (<stdlib.h>)

Esta biblioteca contiene los prototipos de las funciones, macros y tipos para utilidades de uso general.

Función system

Esta función pasa una cadena de texto al entorno local para ser ejecutada por el sistema (comandos de interfaz en modo texto).

Esquema:

```
system(comando);
```

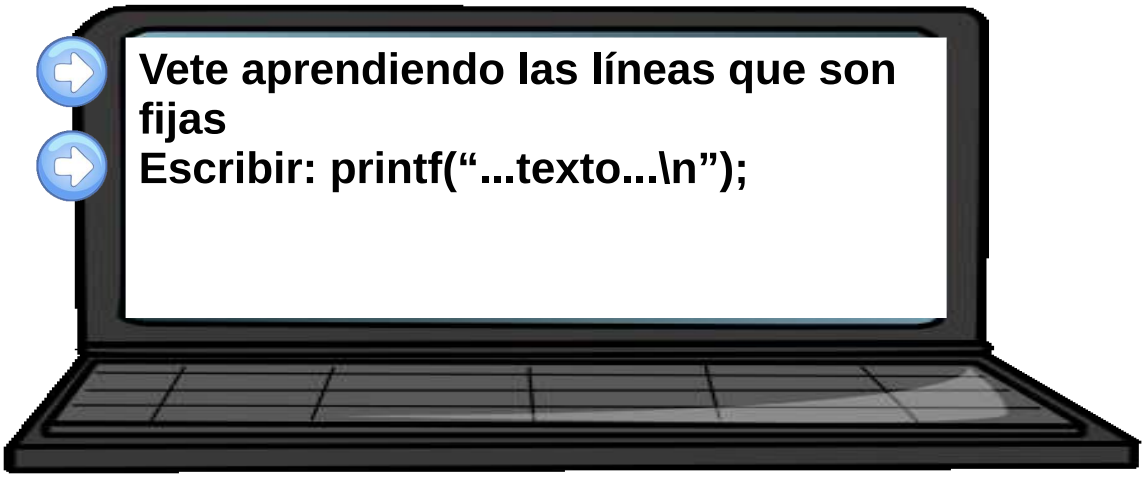
Ejemplo:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* Solo funciona si el comando "dir" es aceptable por el sistema
       :
5  MS-DOS, por ejemplo */
6
7  void main(  )
8  {
9     printf( "La lista de ficheros en el directorio actual, segun
          el comando
10 \"dir\":" );
11    system( "dir" );
12 }
```

Ejemplos resueltos: Escritura

Saludo

```
1 #include <stdio.h>
2
3 void main()
4    {
5    printf("Hola y adios\n");
6    }
```



¿Cómo se manejan valores?

¿Cómo se hacen cálculos?

¿Cómo se leen datos?

¿Cómo se escriben resultados?

C: Constantes

Las constantes son valores fijos durante la ejecución del programa. Se escriben en mayúsculas y se declaran utilizando la directiva `#define` siguiendo la sintaxis:

```
#define NOMBRE valor
```

Ejemplos:

```
1   #define PI 3.141593
2   #define CIERTO 1
3   #define FALSO 0
4   #define AMIGA "Marta"
```

Estilo: Constantes

- Todo valor constante que se use más de una vez deberá estar declarado al principio del programa con `#define`.
- Los nombres usados para constantes seguirán las siguientes reglas:
  - serán sustantivos
  - irán en mayúsculas

Errores frecuentes: Constantes

- Es un error poner ; en instrucciones que empiezan por #
- Es un error usar = en la instrucción `#define`

C: Tipos de datos

Los tipos de datos básicos de C se presentan en la tabla 0.2.

Tipo de dato	Nombre
carácter, letra	char
entero	int
fraccionario (simple precisión)	float

Tabla 0.2: Tipos de datos básicos

C: Declaración de variables

Las variables son datos de un tipo específico que puede cambiar de valor a lo largo del programa. Las variables deben declararse antes de su uso, de esta forma:

```
tipo_de_datos var1, var2,..., varN;
```

Ejemplos:

```
1    int a, b, c;
2    float numero1, numero2;
3    char letra;
```

La conversión entre tipos de datos se realiza por medio del *"casting"*, así:

```
(tipo de dato)  expresión
```

Un ejemplo de utilización:

```
1 int x;
2 float y;
3 x = 5;
4 y = (float)x / 2;
```

Estilo: Declaración de variables

- No usar variables globales salvo absolutamente imprescindible.
- Todas las variables de cada función se declararán al principio de ésta (salvo que sea imposible), luego se pondrá una separación (línea en blanco), y a continuación las instrucciones.
- Los nombres usados para variables seguirán las siguientes reglas:
  - serán sustantivos
  - irán en minúsculas.
- Las variables cuyo sentido no quede suficientemente aclarado con su nombre, tendrán un comentario explicativo junto a su declaración.

C: NOMBRES

Sólo pueden usarse caracteres del alfabeto anglosajón (ni acentos ni eñes).
NO son válidos aquellos que comienzan por:

- Guión -
- Número
- Comilla doble ”

Ejemplos:

Nombres válidos	Nombres NO válidos
area_circulo	-num
dia2	1dia
valor_1	”valor

Tabla 0.3: Ejemplos de nombres

Palabras reservadas:

Las palabras reservadas no pueden ser usadas como nombres.

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

Tabla 0.4: Palabras reservadas: no se pueden usar para nombres

C: UNIDADES O TIPOS DEFINIDOS POR EL PROGRAMADOR

El programador puede definir las unidades de las variables o sus propios tipos de datos con la palabra reservada **typedef** de esta manera:

```
typedef tipo Nuevo_tipo;
```

Esta instrucción la pondremos al inicio del programa, tras los `#include`
Ejemplo:

```
1  typedef  char  Letra;
2  Letra  c;
```

Estilo: Tipos definidos

- Los nombres usados para tipos seguirán las siguientes reglas:
  - serán sustantivos
  - tendrán la inicial mayúscula y el resto irá en minúsculas

Uso

Plantilla:

```
Nuevotipo variable;
```

Ejemplo:

```
1 Metros radio;
```

C: Escritura de variables

Se usa printf, pero hay que indicar en el texto dónde va la variable. En el lugar deseado se pone un código de formato: un % seguido, por lo menos, de una letra que indica el tipo de valor que es (generalmente la inicial) En la tabla 0.5 se muestran las posibilidades.

TIPO DE ARGUMENTO	CARÁCTER DE FORMATO	FORMATO DE SALIDA
Numérico	%i	int
	%f	[-]dddd.dddd
	%e	[-]d.dddde[+/-]ddd (notación científica con potencias de 10)
	%g	el más corto de %e y %f
Carácter	%c	carácter simple
	%s	texto, secuencia de caracteres
	%%	el carácter %

Tabla 0.5: Tabla de conversión de formatos

Plantillas: Escritura de variables

```
printf(" ...%...",variable,...);
```

Ejemplo:

```
1 printf("Es %i\n",horya);
```

Formatos más comunes en la práctica:

%c	una letra
%s	un texto (pág. 171)
%i	un número sin decimales
%f (%.◇f)	un número con decimales (se puede especificar cuántos decimales)

C: Lectura

Función scanf

Esta función lee de teclado y asigna los valores a las variables.

Las variables deben llevar el operador &

Obtiene también la cantidad de variables leídas.
Esquema:

scanf(formato, arg1,. . . , argN);

Ejemplo:

```
scanf("%i",&var);
```

El formato se establece según la tabla de conversión 0.5 de la página 37.

Errores frecuentes: Lectura

- Utilizar formatos equivocados en `scanf`.

Hay que poner & delante de la variable excepto cuando se usa %s

-

Ejemplos resueltos: Lectura

Lectura de un valor y su escritura

```
1 #include <stdio.h>
2
3 void main()
4    {
5    int numero;
6    printf("Pon un número: ");
7    scanf("%i",&numero);
8    printf("El numero era: %i\n", numero);
9    }
```

C: Cálculos

Los cálculos en C más sencillos tendrán la forma: `resultado=calculo;` Hay que fijarse en:

- el resultado tiene que ser una variable, y se pondrá a la izquierda
- luego va el signo =
- luego, a la derecha, el cálculo correspondiente, que serán variables combinadas con operadores

Ejemplo: multiplicar el valor de la variable `base` por el valor de la variable `altura` y meter el resultado en la variable `area` (recordamos que los nombres no pueden llevar acentos) `area=base*altura;`

Recursos en el aula virtual de cálculos simples

Teoría

Aparte de estos apuntes

Ejercicios iniciales de instrucciones concretas

Para comprobar que has interiorizado la teoría
Y recuerda las cuestiones de autoevaluación que tienes aquí en el apartado III

Ejemplos explicados

Con la solución y explicaciones
Y también en estos apuntes, en el apartado 0.2

Ejemplos activos

Visualizaciones de la ejecución paso a paso de un programa

Ejemplos con solución

La solución sin o con pocas explicaciones
Y también en estos apuntes, en el apartado III

Ejercicios con pistas

Sin memorización de por dónde vas

Ejercicios propuestos

Sin la solución

Controles y exámenes de otros años

Algunos con solución, como ejemplos resueltos, y otros sin ella, como ejercicios propuestos

Operadores

Los operadores son los símbolos que indican las operaciones de cálculo que se han de realizar en una expresión. Atendiendo al tipo de operación que realizan se pueden clasificar en:

Supongamos que la variable `etapas` valga 3 y que la variable `marcos` valga 31 y que ambas sean sin decimales.

	OPERADOR	DESCRIPCIÓN	INSTRUCCIÓN EJEMPLO Y RESULTADO
UNARIOS (van delante o detrás de una variable sola)	-	Cambio de signo	`resul= -etapas;` `resul` pasa a valer -3
BINARIOS (van entre dos variables)	-	Resta	`resul= marcos-etapas;` `resul` pasa a valer 28
	+	Suma	`resul= marcos+etapas;` `resul` pasa a valer 34
	*	Producto	`resul= marcos*etapas;` `resul` pasa a valer 93
	/	División	`resul= marcos/etapas;` `resul` pasa a valer 10 (observa que es una operación sin decimales)
	%	Resto de división entera (sólo para enteros)	`resul= marcos % etapas;` `resul` pasa a valer 1

Tabla 0.6: Operadores aritméticos

Operadores aritméticos

En la tabla 0.6 se recogen los operadores aritméticos disponibles en C.
Fíjate que **no hay operador de potencia**; se hace con una función (ver pag. 43)

Operadores de incremento y decremento

En la tabla 0.7 se recogen los operadores incremento (sumar 1) y decremento (restar 1) disponibles en C.

La expresión	es equivalente a
`i++;`	`i = i + 1;`
`++i;`	`i = i + 1;`
`i--;`	`i = i - 1;`
`--i;`	`i = i - 1;`

Tabla 0.7: Operadores incremento y decremento

Operadores de asignación

Son operadores binarios, donde el operando de la izquierda debe ser una variable y el operando de la derecha, un dato del tipo de dicha variable o que pueda ser convertido a dicho tipo. Los operadores de asignación se dividen en:

- Operador de asignación simple, =, ya le hemos visto, que asigna el valor del operando derecho al operando izquierdo. Es decir que `a=b;` quiere decir mirar el valor que ahora mismo tiene `b`

y copiarlo en `a` (lo que hubiese antes en `a` se pierde). No se crea una relación permanente entre las dos variables.

- Operador de asignación múltiple, `op=`, que realizan una operación donde está implicado el operando izquierdo al que es asignado el resultado. En estas expresiones el operando `op` puede ser cualquiera de los siguientes: `*, /, +, -, %`. Es decir, que `a += b;` quiere decir: mira el valor que tienen ahora mismo `a` y `b`, súmalos y pon el resultado en `a` (perdiendo lo que hubiese antes).

Funciones disponibles

Utilidades: (<stdlib.h>)

Esta librería contiene los prototipos de las funciones, macros y tipos para utilidades de uso general.

Función exit . Esta función termina el programa. Todos los ficheros abiertos con datos almacenados aún sin escribir son liberados y todos los ficheros abiertos son cerrados. Si se le llama con el valor cero o EXIT_SUCCESS, se considera que se ha finalizado correctamente; si el valor es EXIT_FAILURE, se considera el caso contrario.

exit(EXIT_FAILURE);

Función system . Esta función ejecuta otro programa o un comando del sistema operativo
Plantilla:

system("programa o comando");

Ejemplo:

```
system(''pause'');
```

Manejo de caracteres: (<ctype.h>)

Esta biblioteca contiene los prototipos de las funciones y macros que permiten la manipulación de caracteres.

Función tolower. Convierte un carácter a minúscula.

min= tolower(ch);

Esta función retorna el valor ch convertido. Si ch está entre A y Z lo convierte a su equivalente en el rango a a z, el resto de los valores no son modificados. Los caracteres acentuados, o con diéresis, en minúscula y la Ñ no sufren modificaciones.

Función toupper. Convierte un carácter a mayúscula.

may = toupper(ch);

Esta función retorna el valor ch convertido. Si ch está entre a y z lo convierte a su equivalente en el rango A a Z, el resto de los valores no son modificados. Los caracteres acentuados, o con diéresis, en mayúscula y la Ñ no sufren modificaciones.

Biblioteca matemática: (<math.h>)

Contiene los prototipos de las funciones matemáticas y otras definiciones para su uso y manipulación.

Función acos. Calcula, en radianes [0, PI], el valor del arco coseno de x. Puede producirse un error para los x que no estén en el intervalo [-1, +1].

Ejemplo:

```
angulo = acos(x);
```

Función asin . Calcula, en radianes [-PI/2, +PI/2], el valor del arco seno de x. Puede producirse un error para los x que no estén en el intervalo [-1, +1].

Ejemplo:

```
angulo = asin(x);
```

Función atan . Calcula, en radianes [-PI/2, +PI/2], el valor del arco tangente de x.

Ejemplo:

```
angulo = atan(x);
```

Función atan2 . Calcula, en radianes [-PI, +PI], el valor del arco tangente de y/x, usando los signos de ambos argumentos para determinar el cuadrante del valor de retorno. Puede producirse un error si ambos son cero.

Ejemplo:

```
angulo = atan2(y, x);
```

Función ceil . Calcula el valor entero más pequeño que no sea menor de x.

Ejemplo:

```
superior = ceil(x);
```

Función cos . Calcula el coseno de x en radianes.

Ejemplo:

```
resul = cos(x);
```

Función cosh . Calcula el coseno hiperbólico de x. Puede producirse un error si la magnitud de x es demasiado grande.

Ejemplo:

```
resul = cosh(x);
```

Función exp. Calcula la función exponencial de x, es decir, e^x.

Ejemplo:

```
resul = exp(x);
```

Función fabs. Calcula el valor absoluto del número con decimales x.
Ejemplo:

```
sindec = fabs(x);
```

Función floor . Calcula el valor entero más grande que no sea mayor de x.
Ejemplo:

```
inferior = floor(x);
```

Función fmod . Calcula el resto de la división de x/y que son con decimales.
Ejemplo:

```
resto = fmod(x, y);
```

Función log . Calcula el logaritmo natural o neperiano, es decir, $\ln(x)$. Se produce un error si x es negativo o cero.
Ejemplo:

```
resul = log(x);
```

Función log10 . Calcula el logaritmo en base 10 de x, es decir, $\lg(x)$. Se produce un error si x es negativo o cero.
Ejemplo:

```
resul = log10(x);
```

Función pow . Calcula x elevado a la potencia de y, es decir, x^y. Puede producirse un error si x es negativo e y no es un valor entero. También se produce un error cuando x es cero e y es menor o igual que cero.
Ejemplo:

```
potencia = pow(x, y);
```

Función sin . Calcula el seno de x en radianes.
Ejemplo:

```
seno = sin(x);
```

Función sinh . Calcula el seno hiperbólico de x. Se produce un error si la magnitud de x es demasiado grande.
Ejemplo:

```
resul = sinh(x);
```

Función sqrt . Calcula la raíz cuadrada del valor no negativo de x. Puede producirse un error si x es negativo.
Ejemplo:

```
raiz = sqrt(x);
```

Función tan . Calcula la tangente de x en radianes.
Ejemplo:

```
valor = tan(x);
```

Función tanh . Calcula la tangente hiperbólica de x.
Ejemplo:

```
valor = tanh(x);
```

Utilidades: (<stdlib.h>)

Esta librería contiene los prototipos de las funciones, macros y tipos para utilidades de uso general.

Función abs . Calcula el valor absoluto del argumento entero num.
Ejemplo:

```
sinsig = abs(num);
```

Función srand . La función srand se usa para preparar la generación de una secuencia de números pseudo-aleatorios. Si srand se llama con el mismo valor, la secuencia de números pseudo-aleatorios será repetida. Si no se usa srand es equivalente a usarla con un valor 1.
Ejemplo típico (copia y pégalo cuando necesites números aleatorios):

```
srand(time(NULL));
```

Función rand . La función rand saca un valor pseudo-aleatorios en el intervalo de 0 á RAND_MAX, que es al menos 32767. En uno de los ejemplos de listas (pág. 159) aparece utilizada.
Ejemplo:

```
aleat = rand();
```

Fecha/hora: <time.h>

Prepara las funciones, macros y tipos para manipular la hora y la fecha del sistema. Define, entre otros, los tipos *clock_t* y *time_t* como representaciones del tiempo en un único número.

Función time. La función time determina el tiempo en formato `time_t`. Si el tiempo no está disponible, la función retorna el valor -1.
Ejemplo-plantilla:

```
tiempotime_t=time(NULL);
```

Función ctime. La función *ctime* convierte el tiempo recogido con la función anterior, en un texto de la forma: Tue May 15 19:07.04 2001\n. Se usa típicamente para escribir la fecha y hora en pantalla.
Plantilla:

```
printf(‘‘%s’’,ctime(time(NULL));
```

Puede pasársele una variable recogida como resultado de la función time.

Función clock. La función *clock* determina el tiempo usado del procesador desde el inicio del programa. Para determinar el tiempo en segundos, el valor retornado por la función *clock* debería ser dividido por el valor CLOCKS_PER_SEC. Si el tiempo usado del procesador no está disponible o su valor no puede ser representado, la función retorna el valor -1.

Ejemplo:

mide=clock();

Ejemplo de uso

```
#include <stdio.h>
   #include <time.h>

   int main( void )
   {
      int i=0;
      time_t comienzo, final;

      comienzo = time( NULL );
      for( i=0; i<10000; i++ )   printf( "-" );
      final = time( NULL );

      printf( "Comienzo: %u s\n", comienzo );
      printf( "Final: %u s\n", final );
      printf( "Segundos desde el comienzo del programa: %is\n",
       final-comienzo );


      return 0;
   }
```

Plantilla típica de instrucción de cálculo

variable resultado= expresión de cálculo;

Siempre va la variable donde se pone el resultado a la izquierda del igual.

A la derecha del igual se pone la expresión de cálculo Y siempre un ; al final

Ejemplo: sacar la raíz cuadrada del doble de lo que haya en `ancho` y poner el resultado en `perdida`: perdida=sqrt(ancho*2);

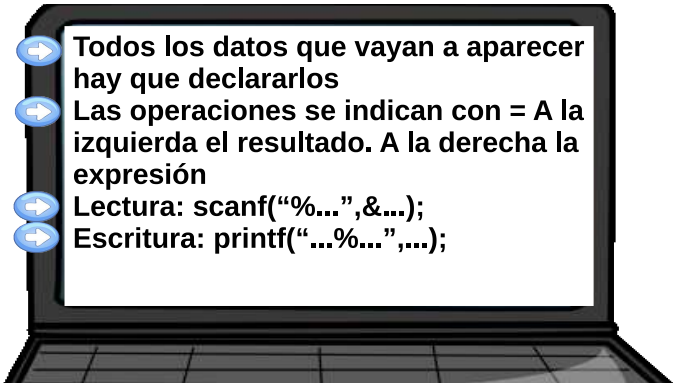
Ejemplo: elevar la suma de `base1` más `base2` al exponente resultado de restar `coef1` y `caso`, poniendo el resultado en `potencia`: potencia=pow(base1+base2,coef1-caso);

Estilo: Cálculos

Las expresiones matemáticas muy complejas se pondrán en varios pasos.

Ejemplos de instrucciones

Aquí se presentan ejemplos de instrucciones de lo que hemos visto.




```
saldo -= dispo;
   printf("Aviso tipo %i\n",avtip);
   hip = sqrt(sumcuad);
   Cantidad pedido1;
   printf("Opcion %c\n",letop);
   printf("%i-%i final\n",tantos1,tantos2);
   typedef float Areas,Hectareas;
   scanf("%f",&tiempo);
   printf("Error fatal\n");
   cuenta++;
   #define VALADM 500
   #define TERMIN '0'
   Litros vaso, botella;
   typedef int Personas;
   int cuenta, final;
   typedef float Julios;
   printf("Volumen: %.1f\n",volcubo);
   area=base*altura;
   acum=0;
   scanf("%f %i",&final,&tipo);
   char peticion;
   scanf("%i",&num);
   #define GAMMA 1.03
```

Cuestiones de autoevaluación

Preguntas

1-Dar la instrucción C que lee dos números en las variables con decimales a y b, sólo con el espacio imprescindible. Usar el código de formato f.

2-En C seno es sin. ¿Cómo es coseno?

3-En C el logaritmo neperiano es log. ¿Cómo es el logaritmo decimal?

4-¿Cuántas variables independientes se pueden leer con un solo scanf?
a-1
b-2
c-3
d-las que nos venga bien

5-Dar, sin espacios, la instrucción C que asigna a la variable x el logaritmo neperiano dela suma de su valor anterior y el de la variable z (ambas son con decimales).

6-Dar la instrucción que declara la constante PI con el valor 3.1416. Poner sólo espacios imprescindibles.

7-Volver a escribir esta instrucción de forma correcta (con los mismos espacios)#define E 2,71828

8-En C el valor absoluto de números sin decimales se pone abs. ¿Cómo es el valor absoluto de números con decimales?

9-Escribir la instrucción C que pone en la variable x el resultado de elevar el valor de y al valor de n+1. Todas son con decimales. No poner espacios.

10-Para usar las funciones trigonométricas hay que poner al principio del programa
#include <_____.h>
¿Qué va en el hueco?

11-Escribir la instrucción C que pone en la variable x el resultado de la expresión PI(r+INC). Todas son con decimales. No poner espacios.

12-Si a vale 2, b vale 4 y c vale 0.5 ¿cuanto vale la siguiente expresión C?
pow(a,pow(b,c))

13-Dar la instrucción C que pone en resul el cociente entre dat1 y la resta dat1-dat2. No poner espacios.

14-Una instrucción que sume 1 a la variable x puede ser:
x=x+1;
También puede ser:
x+=1;
y también puede ser....

15-Se tiene en un programa la instrucción
#define PI 3.1416

Se decide dar la oportunidad al usuario de cargar otro valor para PI, con la instrucción:
a-scanf("%f",&PI);
b-scanf("%f",PI);
Contestar a,b,ab o ninguna

16-¿Cuántas veces aparece la palabra main en un programa en C (sin contar posibles comentarios)? Dar una contestación numérica.

17-Dar una instrucción C que meta en la variable sufijo el valor formado con las últimas tres cifras de la variable numero. No poner espacios. Usar la constante MIL, que tiene el valor 1000.

18-¿Qué función C es la opuesta al logaritmo neperiano, e elevado a un valor?

19-Se quiere poner en la variable y el resultado de elevar el valor absoluto de x-m al exponente a, usando una sola instrucción. Todas las variables tienen decimales. No poner espacios.

20-La tangente en C es tan. ¿Cómo es el arco tangente? Dar las dos posibilidades separadas por un espacio, primero la de menos letras

21-Se tiene una diana con círculos cuyo radio va de 1 en 1. El más interior puntúa 10 y el resto decrece de 1 en 1. Si se tiene la distancia de un dardo al centro en la variable distan (con decimales), la instrucción que pone en la variable puntos (sin decimales) la puntuación obtenida acaba en ...(int)distan. ¿Qué va delante? No poner espacios.

22-Dar la instrucción que pone en la variable numcif (sin decimales) cuántas cifras tiene el número sin decimales que está en la variable numero (con decimales, aunque no se usan).

23-Escribir en una sola instrucción un mensaje en que aparezcan los valores de las variables a y b separados por un + y después el signo = y su suma y finalmente un salto de línea. Todas las variables son sin decimales (usar el formato %i). No poner espacios.

24-Se separa la parte entera de un valor positivo con decimales pasándolo a una variables sin decimales, como:
ent=numero;
A continuación se quiere poner en la variable frac la parte fraccionaria. Dar la instrucción correspondiente usando las variables anteriores y sin espacios.

25-Se pone la instrucción siguiente:

```
if (scanf("%i",&num)==__)
```
Queremos que al ejecutar, si el usuario teclea una letra la condición sea cierta. ¿Qué hay que poner en el hueco?

26-Se va a trabajar con variables que viene en metros. Posiblemente tendrán decimales. Dar la instrucción que define este nuevo tipo (usar para su nombre la palabra completa). Poner sólo los espacios imprescindibles.

27-A una variable la llamamos cañí. Escribir, seguidas y por el orden en que aparecen, las letras que el compilador no admitirá.

28-Se tiene el siguiente fragmento de programa
```
/* Aqui calcula el area
area=lado*lado;
```
Escribir los dos caracteres que faltan, por el orden que hay que ponerlos.

29-Dar las letras, seguidad y por el orden presentado, de los nombres de variables válidos
a) y12
b) 4num
c) suma_1
d) orden-no

30-Ordenar de mejor a peor los nombres de variables elegidos para guardar un precio
a) pr
b) precio
c) x
Escribir las letras sin separación

31-Dar, por el orden presentado, los formas válidas de escribir en C trescientos mil
a) 300000,00
b) 300000
c) 3e 5
d) 3e5
Escribirlas sin separación

32-Se quiere que aparezca en pantalla las siguientes dos líneas:
```
      Es "hola"
      En 'c:\'
```
Se usa la instrucción printf("...."); ¿qué va en el hueco?

33-Se quiere en un programa que las variable p sea un número sin decimales, x sea con decimales y a sea una letra. Dar las declaraciones todas seguidas en la misma línea con sólo los espacios

imprescindibles.

34-Dar los números resultados de las siguientes expresiones por el orden pedido y separados por espacios. Si alguna es incorrecta poner error. La variable i es sin decimales y vale 7. La variable f es con decimales y vale 8.5

```
i+f
(i+f)%4
i-f
```

35-Se quiere escribir el valor de la variable con decimales x con 4 decimales. Se pone:

```
printf("%____f",x);
```

¿Qué va en el hueco?

36-La función floor redondea un número con decimales hacia abajo. ¿Cuál es la que lo redondea hacia arriba?

Respuestas

1 `scanf("%f %f",&a,&b);`

2 cos

3 log10

4 d

5 `x=log(x+z);`

6 `#define PI 3.1416`

7 `#define E 2.71828`

8 fabs

9 `x=pow(y,n+1);`

10 math

11 `x=PI*(r+INC);`

12 4

13 `resul=dat1/(dat1-dat2);`

```
14 x++;
15 ninguna
16 1
17 sufijo=numero%MIL;
18 exp
19 y=pow(fabs(x-m),a);
20 atan atan2
21 puntos=10-(int)distan;
22 numcif=log10(numero)+1;
23 printf("%i*%i=%i\n",a,b,a+b);
24 frac=numero-ent;
25 0
26 typedef float Metros;
27 ñí
28 */
29 ac
30 bac
31 bd
32 Es \"hola\"\nEn \'c:\\\'
33 int p;float x;char a;
34 15.5 error -1.5
35 .4
36 ceil
```

Ejemplos resueltos: Calcular

Cálculo de área y perímetro de circunferencia

Enunciado

Programa que lea el radio de una circunferencia y escriba su área y su perímetro.

Solución

Vamos a proceder en dos pasos. En el primero desarrollaremos el esquema del programa en nuestro idioma (esto suele llamarse seudocódigo), y luego lo traduciremos a C

Pasos del programa establecidos en el enunciado.

1. que lea el radio
2. escriba su área
3. y su perímetro

Completar los pasos. El primero se puede hacer directamente, pero el segundo y el tercero no, porque no conocemos el área y el perímetro, sino el radio. Habrá que calcularlos. Entonces nos queda:

1. que lea el radio
2. calcular el área
3. calcular el perímetro
4. escriba su área
5. y su perímetro

Variables (valores) que aparecen. radio, área, perímetro

Detalle de los pasos de cálculo. En todo cálculo hay que tener claro:

- Qué operaciones concretas hay que hacer
- En qué variable queda el resultado

En nuestro caso, si detallamos los pasos de cálculo nos queda:

1. que lea el radio
2. calcular el área
   - Operaciones: $\pi \cdot radio^2$
   - Resultado: área
3. calcular el perímetro
   - Operaciones: $2 \cdot \pi radio$
   - Resultado: perímetro

4. escriba su área
5. y su perímetro

Y ya tenemos el esquema o seudocódigo completo

Traducción a C.

que lea el radio. Siempre que leamos algún dato de teclado, haremos dos cosas:

1. Poner un mensaje pidiendo el dato
2. Recoger lo que se teclee en una variable

Para lo primero es la instrucción `printf` (pág. 31). Nuestro mensaje puede ser algo como *¿Cuál es el radio?* Queda: `printf("¿Cuál es el radio? ");`

Para l segundo es la instrucción `scanf` (pág. 38). A la variable la vamos a llamar `radio` y pensamos que puede tener decimales, luego nos queda: `scanf(" %f",&radio);`

calcular el área. Teniendo en cuenta

1. los detalles de este cálculo
2. cómo se organizan las instrucciones de cálculo (pág. 39)
3. cómo se escriben las operaciones (pág. 40)
4. a π lo llamaremos `PI`
5. el cuadrado lo vamos a calcular multiplicando el radio por sí mismo
6. a la variable para el área la llamaremos `area` porque no puede llevar acentos (pág. 36)

nos queda `area=PI*radio*radio;`

calcular el perímetro. Es casi igual; a la variable para el perímetro la llamaremos `perimetro`, así que nos queda `perimetro=2*PI*radio;`

escriba su área. La variable es `area` (tendrá decimales) y la instrucción para esto es `printf` (pág. 37). Terminaremos la línea (pág. 31) para que el siguiente mensaje salga en otra, así que nos queda `printf("El área es %f\n",area);`

y su perímetro. Análogamente: `printf("El perímetro es %f\n"perimetro);`

Con eso tenemos ya traducidas todas las instrucciones, la operativa del programa, pero todavía tenemos que considerar las siguientes cuestiones:

1. ¿Hacemos uso de algunas funciones de C más allá de las básicas?
2. ¿Nuestras variables están en algún tipo de unidades? ¿De qué tipo son?
3. ¿Necesitamos alguna constante?

Vamos con ello

Funciones de C. Sólo usamos lo básico, así que nada que añadir por ahí

Variables y unidades. Nos salen longitudes y áreas, así que sí tendremos unidades. Vamos a pensar que trabajamos en el Sistema Internacional. El desglose por variable nos queda:

Variable	Unidades (nombre en C)	Tipo (pág. 35)
radio	Metros	float
area	Metros2	float
perimetro	Metros	float

De donde sacamos la siguiente instrucción para las unidades (pág. 36) `typedef float Metros, Metros2;`

Y la declaración de variables (págs. ?? y 36) nos queda:

```
1         Metros radio,perimetro;
2         Metros2 area;
```

Constantes. Necesitamos π Pensemos que con cuatro decimales sea suficiente. Nos queda (pág. 34) `#define PI 3.1416`

Composición final del programa. Con todo lo anterior, teniendo en cuenta cómo se colcan las distintas partes de un programa (págs. 30, 36 y ??), añadiendo un comentario explicativo, nos queda la solución final:

```
1  #include <stdio.h>
2  #include <locale.h>
3  typedef float Metros, Metros2;
4  #define PI 3.1416
5
6  /* Este programa calcula el área y perímetro de un círculo a
      partir del radio */
7
8  void main () {
9          Metros radio,perimetro;
10         Metros2 area;
11
12         setlocale(LC_CTYPE,"");
13         printf("¿Cuál es el radio? ");
14         scanf(" %f",&radio);
15         area=PI*radio*radio;
16         perimetro=2*PI*radio;
17         printf("El área es %f\n",area);
18         printf("El perímetro es %f\n"perimetro);
19 }
```

Calcular área de rectángulo

```
1 #include <stdio.h>
2 typedef float Metros, Metros2;
3 /*~TypeCombination
4 Metros * Metros -> Metros2 */
```

```
5
6  void main()
7     {
8     Metros base, altura;
9     Metros2 area;
10    printf("Dar base y altura: ");
11    scanf("%g %g",&base, &altura);
12    area = base * altura;
13    printf("El área es: %g\n", area);
14    }
```

Calcular área de trapecio

```
1  #include <stdio.h>
2  /* La fórmula es: base media por la altura */
3  typedef float Metros, Metros2;
4  /*~TypeCombination
5  Metros * Metros -> Metros2 */
6
7  void main()
8     {
9     Metros base1, base2, altura;
10    Metros2 area;
11    printf("Dar bases y altura: ");
12    scanf("%g %g %g",&base1, &base2, &altura);
13    area = ((base1 + base2) / 2) * altura;
14    printf("El área es: %g\n", area);
15    }
```

En este caso los paréntesis se utilizan para controlar el orden de operación:

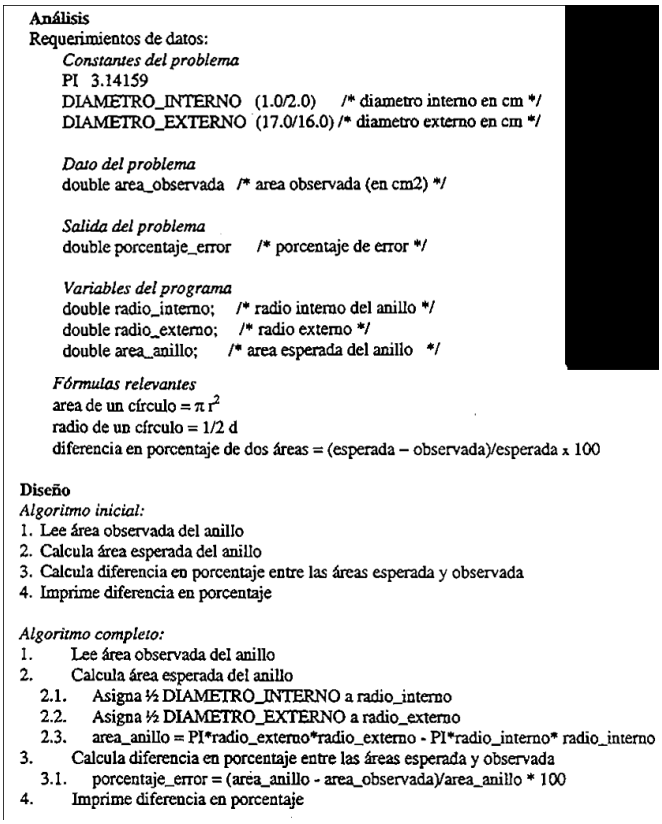
1. Sumar las dos bases
2. Dividir por 2
3. Multiplicar por la altura

Cálculo de área de anillo circular

En las figuras 0.2, 0.3 y 0.4 está respectivamente el enunciado, razonamiento y desarrollo del programa. Está tomado del cuaderno de ejercicios del profesor Pedro Corcuera.

Con objeto de automatizar un proceso de producción, se instala un sistema que calcula el área de un anillo. Se pide elaborar un programa para calcular la diferencia entre el área observada y esperada y el error relativo de un anillo respecto de uno que sirve como patrón (diametro interno= ½ cm., diámetro externo=17/16 cm.).

Figura 0.2: Enunciado de programa para calcular el área de un anillo circular



Análisis
Requerimientos de datos:
Constantes del problema
PI 3.14159
DIAMETRO_INTERNO (1.0/2.0) /* diametro interno en cm */
DIAMETRO_EXTERNO (17.0/16.0) /* diametro externo en cm */

Dato del problema
double area_observada /* area observada (en cm2) */

Salida del problema
double porcentaje_error /* porcentaje de error */

Variables del programa
double radio_interno; /* radio interno del anillo */
double radio_externo; /* radio externo */
double area_anillo; /* area esperada del anillo */

Fórmulas relevantes
area de un círculo = πr^2
radio de un círculo = 1/2 d
diferencia en porcentaje de dos áreas = (esperada – observada)/esperada x 100

Diseño
Algoritmo inicial:
1. Lee área observada del anillo
2. Calcula área esperada del anillo
3. Calcula diferencia en porcentaje entre las áreas esperada y observada
4. Imprime diferencia en porcentaje

Algoritmo completo:
1. Lee área observada del anillo
2. Calcula área esperada del anillo
   2.1. Asigna ½ DIAMETRO_INTERNO a radio_interno
   2.2. Asigna ½ DIAMETRO_EXTERNO a radio_externo
   2.3. area_anillo = PI*radio_externo*radio_externo - PI*radio_interno* radio_interno
3. Calcula diferencia en porcentaje entre las áreas esperada y observada
   3.1. porcentaje_error = (area_anillo - area_observada)/area_anillo * 100
4. Imprime diferencia en porcentaje

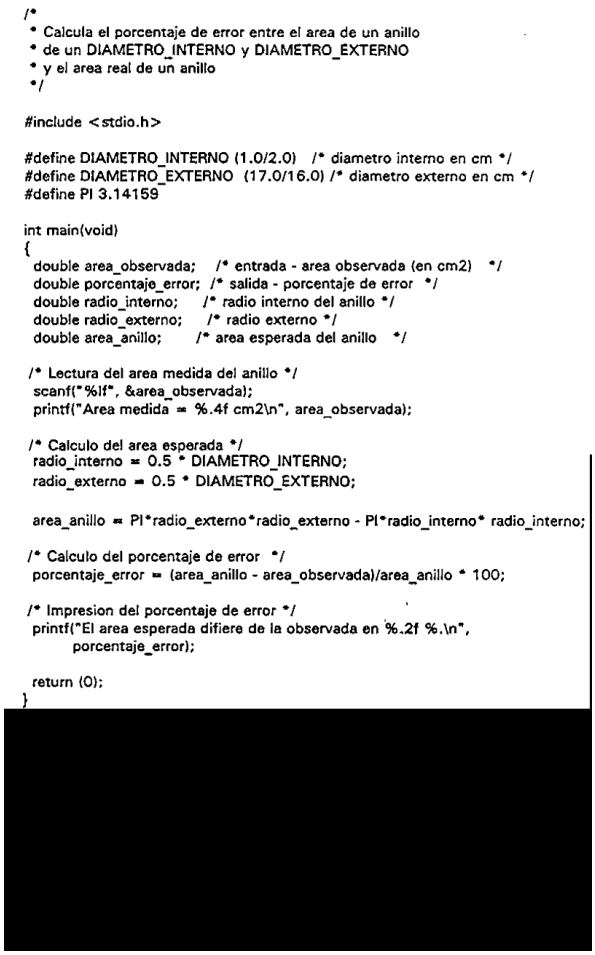
Figura 0.3: Deducción de las instrucciones necesarias para calcular el área de un anillo

Arco coseno

```
1 #include <stdio.h>
2 #include <math.h>
3
4 void main()
5 {
6    double x = 0.2345;
7
8    printf( "acos( %f ) = %f\n", x, acos(x) );
9 }
```

Calcular volumen de esfera

```
1 #include <stdio.h>
2 #include <math.h> /* Para la función pow, que eleva a potencia
     */
3 #define PI 3.1416
4 /* La fórmula es: cuatro tercios de pi radio al cubo
5 El cuatro se pone con punto para que opere con decimales */
6 #define FACTOR (4./3)
7 #define CUBO 3.
8 typedef float Metros, Metros3;
```



```
/*
 * Calcula el porcentaje de error entre el area de un anillo
 * de un DIAMETRO_INTERNO y DIAMETRO_EXTERNO
 * y el area real de un anillo
 */

#include <stdio.h>

#define DIAMETRO_INTERNO (1.0/2.0)   /* diametro interno en cm */
#define DIAMETRO_EXTERNO  (17.0/16.0) /* diametro externo en cm */
#define PI 3.14159

int main(void)
{
  double area_observada;   /* entrada - area observada (en cm2)   */
  double porcentaje_error; /* salida - porcentaje de error  */
  double radio_interno;    /* radio interno del anillo */
  double radio_externo;    /* radio externo */
  double area_anillo;      /* area esperada del anillo   */

 /* Lectura del area medida del anillo */
  scanf("%lf", &area_observada);
  printf("Area medida = %.4f cm2\n", area_observada);

 /* Calculo del area esperada */
  radio_interno = 0.5 * DIAMETRO_INTERNO;
  radio_externo = 0.5 * DIAMETRO_EXTERNO;

  area_anillo = PI*radio_externo*radio_externo - PI*radio_interno* radio_interno;

 /* Calculo del porcentaje de error  */
  porcentaje_error = (area_anillo - area_observada)/area_anillo * 100;

 /* Impresion del porcentaje de error */
  printf("El area esperada difiere de la observada en %.2f %.\n",
         porcentaje_error);

  return (0);
}
```

Figura 0.4: Programa solución para calcular el área de un anillo

```
9
10 void main()
11    {
12    Metros radio;
13    Metros3 volumen;
14    printf("Dar radio: ");
15    scanf("%g",&radio);
16    volumen = FACTOR * PI * pow(radio, CUBO);
17    printf("El volumen es: %g\n", volumen);
18    }
```

Hay que usar la función `pow`, ya que no hay operador de elevar a potencia. En FACTOR se ha puesto un punto decimal para que opere con decimales, si no, el resultado sería 1. El valor de π no es conocido en el C habitual.

Errores frecuentes: Cálculos

Errores frecuentes:

- Paréntesis no equilibrados

- No poner ; en instrucciones normales (asignación)
- Problemas con la precedencia (orden de ejecución) entre operadores. Nunca se producirá un error de compilación. El consejo es usar siempre paréntesis, a no ser que estemos **absolutamente** seguros.

¿Cómo se hace para leer o escribir de un fichero en vez de pantalla?

C: Ficheros

Para trabajar con un fichero en un programa C, hay que seguir, normalmente, los siguientes pasos:

1. Declarar una variable del tipo especial `FILE *` para el fichero. Si por ejemplo la variable fuera a llamarse `fisal`, pondríamos al principio del programa: `FILE *fisal;`

2. Preparar el fichero, usando la función `fopen`, con dos argumentos: el nombre del fichero (puede ser como texto directamente entre comillas o con una variable, que veremos más adelante) y si lo queremos para leer (`"r"`) o para escribir (`"w"`). Si por ejemplo la variable para el fichero se llama `fisal`, tenemos el nombre guardado en la variable `nomfi` y queremos escribir en él, sería `fisal=fopen(nomfi,"w");`

3. Usarlo para leer o escribir, con las funciones `fprintf` o `fscanf`, que son casi iguales que `printf` y `scanf`; sólo cambia que hay que poner un primer argumento extra que es la variable del fichero. Si por ejemplo esa variable es `fisal` y queremos escribir un resultado que tenemos en la variable `resul`, podría ser `fprintf(fisal,"El resultado es %f\n",resul);`

4. Cerrar el fichero con `fclose` (no hace falta si es al final del programa). Lleva sólo un argumento que es la variable del fichero. Por ejemplo, si la variable del fichero es `fisal`, sería `fclose(fisal);`

Veamos más en detalles las funciones implicadas.

Función fopen

Esquema:

fichero=fopen(nombre,modo);

Ejemplo:

```
fichero=fopen("datos.txt","r");
```

Esta función abre un fichero cuyo nombre le pasamos y devuelve una variable especial de tipo `FILE *` que controla el fichero. El modo de trabajo con el fichero es un texto de entre los siguientes:

Si se intenta abrir un fichero en modo lectura (’r’ como primer carácter del argumento modo) fallará si el fichero no existe o no puede ser leído. Si se abre un fichero en modo añadir (’a’) hace que todas las escrituras se realicen al final de fichero (EOF) actual. Cuando un fichero se abre con ’+’ como segundo carácter del argumento modo, se puede hacer entrada y salida.

"r"	Abre un fichero de texto para lectura
"w"	Trunca, a longitud cero o crea un fichero de texto para escribir
"a"	Añade; abre o crea un fichero de texto para escribir al final del fichero (EOF)
"r+"	Abre un fichero de texto para lectura y escritura
"w+"	Trunca a longitud cero, o crea, un fichero de texto para lectura y escritura
"a+"	Añade; abre o crea un fichero de texto para lectura y escritura, poniéndose al final del fichero (EOF)

Tabla 0.8: Modificadores de apertura de ficheros

Función fprintf

Esta función es exactamente análoga a `printf`, pero para escritura en ficheros (o "streams"). Convierte y escribe la salida bajo el control de formato.

Esquema:

fprintf(stream,formato, arg1,..., argN);

Ejemplo:

```
fprintf(fichero,"Esto es un fichero");
```

Ejemplo en programa:

```
1  #include <stdio.h>
2  void main()
3  {
4  FILE *fichero;
5  unsigned int i;
6  fichero = fopen( "datos.dat", "w" );
7  printf( "Fichero: datos.dat (para escritura) -> ");
8  if( fichero != NULL)
9    printf( "Creado (ABIERTO)\n" );
10 else
11 {
12   printf( "Error (NO ABIERTO)\n" );
13   return;
14 }
15 fprintf( fichero, "Esto es un ejemplo de uso de la funcion \'
     fprintf\'\n" );
16 fprintf( fichero, "1\t 2\t 3\t 4\n" );
17 fprintf( fichero, "x\tx\tx\tx\n\n" );
18 fprintf( stdout, "Datos guardados en el fichero: datos.dat\n");
19 }
```

Función fscanf

Esta función es análoga a `scanf` pero para ficheros (o "streams"). Lee bajo el control de formato y asigna los valores a los argumentos, de la misma forma que `scanf`. Retorna el número de argumentos leídos si la operación se realiza con éxito y EOF en caso contrario.

Esquema:

fscanf(stream,formato, arg1,. . . , argN);

Ejemplo:

```
fscanf(fichero,"%i",&var);
```

El formato se establece según la tabla de conversión 0.5 de la página 37.

Función fflush

Si se le pasa un fichero de salida o de actualización cuya operación más reciente no era de entrada, la función `fflush` envía cualquier dato aún sin escribir a ser escrito en el fichero; en un fichero de entrada, el comportamiento no está definido. Si es un puntero nulo, la función `fflush` libera todos los ficheros de salida.

Esquema:

fflush(stream);

Ejemplos resueltos: Ficheros

Calcular dirección absoluta

En la figura 0.5 se explica el programa que calcula la dirección absoluta de una instrucción, conociendo su relativa y las relativa y absoluta de otra instrucción que esté en la misma página.

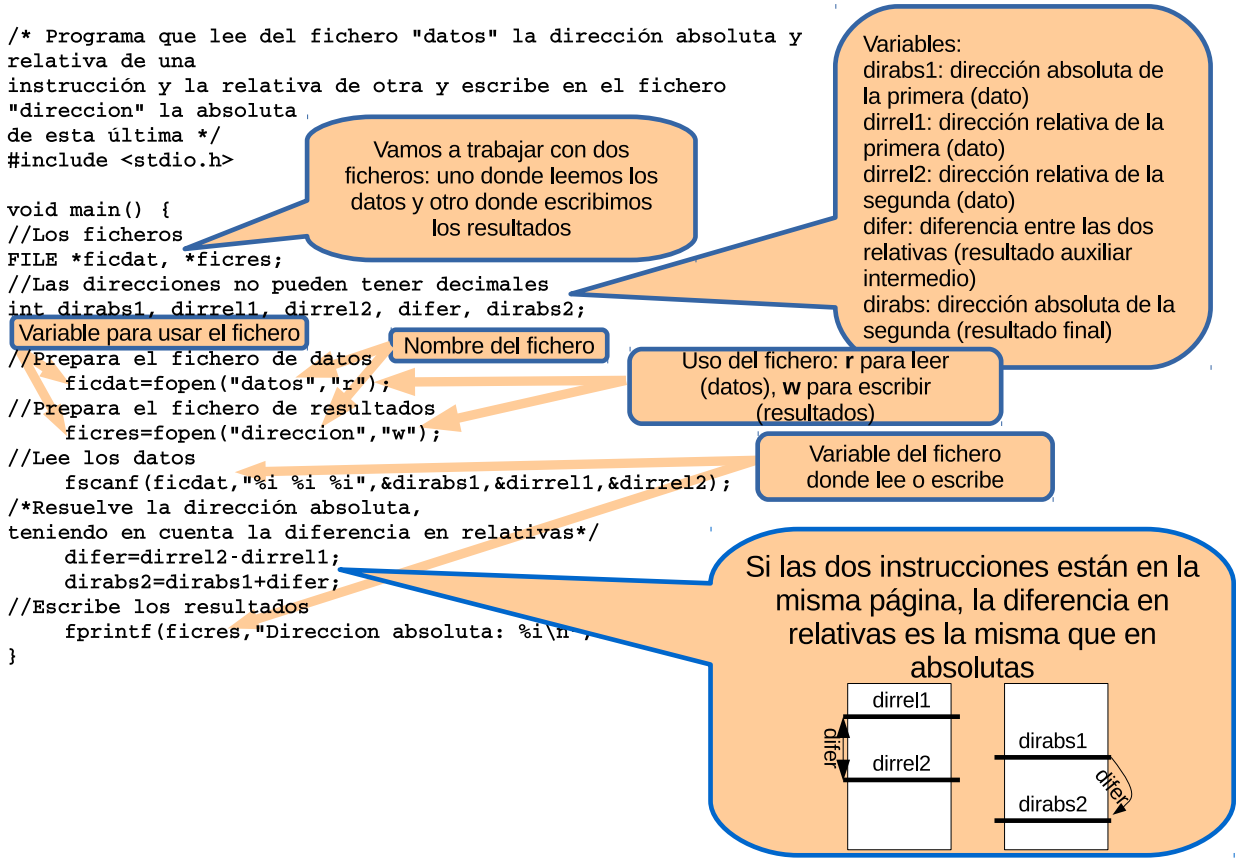


Figura 0.5: Programa para calcular dirección absoluta

Control de flujo

Se llaman instrucciones de control de flujo a las que sirven para decir si otras instrucciones deben ejecutarse o no. En este grupo están las condicionales, que permiten elegir una sola vez entre uno o varios grupos de instrucciones, los ciclos, que permiten repetir varias veces un bloque de instrucciones, y las funciones, que las organizan en bloques y establecen cuándo pasar de un bloque a otro.

C: Condicionales

¿Cómo se marcan distintos casos que llevan distintas instrucciones?

Recursos en el aula virtual de condicionales

Teoría

Aparte de estos apuntes

Ejercicios iniciales de instrucciones concretas

Para comprobar que has interiorizado la teoría
Y recuerda las cuestiones de autoevaluación que tienes aquí en el apartado III

Ejemplos explicados

Con la solución y explicaciones
Y también en estos apuntes, en el apartado III

Ejemplos activos

Visualizaciones de la ejecución paso a paso de un programa
Y también en estos apuntes, en la figura 0.6

Ejemplos con solución

La solución sin o con pocas explicaciones
Y también en estos apuntes, en el apartado III

Ejercicios con pistas

Sin memorización de por dónde vas

Ejercicios propuestos

Sin la solución

Controles y exámenes de otros años

Algunos con solución, como ejemplos resueltos, y otros sin ella, como ejercicios propuestos

Los condicionales nos salen en todos aquellos programas en que hay diferentes casos.

Las sentencias condicionales son aquellas que permiten ejecutar un bloque de instrucciones sólo si se cumple una condición.

Sentencia if

Permite la ejecución de una sentencia o bloque de sentencias si se cumple la condición que lleva asociada.

```
if (condición) {
       instrucciones;
   }
```

Sentencia if-else

Similar a la sentencia `if` pero en caso de que no se cumpla la condición, ejecuta un segundo grupo.

```
if (condición) {
       instrucciones1;
   }
   else {
       instrucciones2;
   }
```

Condiciones

Para las condiciones de la instrucción `if` y, en general, para cualquier condición que se quiera usar en C, se usan operadores relacionales (de comparación), lógicos (de combinar comparaciones) y algunas funciones para caracteres.

Operadores relacionales

En el cuadro 0.9 se recogen los operadores relacionales o de comparación disponibles en C.

OPERADOR	DESCRIPCIÓN
>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor o igual que
==	Igual que (¡Atención!, son dos signos =)
!=	Diferente que

Tabla 0.9: Operadores relacionales

Operadores lógicos

En el cuadro 0.10 se recogen los operadores lógicos (combinar comparaciones) disponibles en C.

OPERADOR	DESCRIPCIÓN
!	no
&&	y (se deben cumplir ambas comparaciones)
||	o (basta que se cumpla una)

Tabla 0.10: Operadores lógicos

Funciones para caracteres

Manejo de caracteres: (<ctype.h>). En caso de condiciones con caracteres pueden ser útiles una serie de funciones que trae C y que vamos a comentar.

Ejemplo de uso (es análogo para todas):

```
if (isalpha(c)) {
```

Macro isalpha. Comprueba si el carácter `c` es alfabético. Da cierto si es una letra y falso en caso contrario.

isalpha(c)

Macro isdigit. Comprueba si el carácter es un dígito decimal, es decir, una cifra de 0 a 9.

isdigit(c)

Macro isgraph. Verifica que `c` pertenece al rango de caracteres con representación gráfica, que por defecto son todos menos el espacio.

isgraph(c)

Macro islower. Comprueba si el carácter es de tipo minúscula, que por defecto son los que están en el rango `a` a `z`, es decir, que no reconoce ni vocales acentuadas ni eñes.

islower(c)

Macro ispunct. Comprueba si el carácter es un signo de puntuación.

ispunct(c)

Macro isspace. Comprueba si el carácter es de tipo espacio, o sea, que pertenece al grupo de caracteres: espacio, ' ', tabulador, retorno de carro, nueva línea, tabulador vertical o salto de página.

isspace(c)

Macro isupper. Comprueba si el carácter es de tipo mayúscula, que por defecto son los que están en el rango A a Z, es decir, que no reconoce ni vocales acentuadas ni Ñ.

isupper(c)

Condiciones múltiples

Cuando hay que comprobar el valor de cierta variable y según cuál sea hay que hacer unas cosas u otras, existen dos posibilidades:

- Una serie de `if` consecutivos, siguiendo el patrón:

```
if (expresion==caso1) {
          instrucciones para el caso1;
   } else if (expresion==caso2) {
           instrucciones para el caso2;
   }         ...
           ...
   }        else {
       instrucciones para resto de casos;
       }
```

- Usar la instrucción `switch`, que se comenta a continuación

Ambas soluciones son correctas, por lo que `switch` (que no vale para cualquier cosa, tiene sus restricciones) es opcional. Si no sabes nada de ella, puede que algún programa te sea algo más incómodo, pero todo podrás hacerlo.

Instrucción ***`switch`***

Ejecuta un bloque de instrucciones u otro, dependiendo del valor de una expresión. Dicha expresión sólo puede ser de tipo entero o carácter.

```
switch (expresion) {
          case cte1:  instrucciones;
              break;
          case cte2:  instrucciones;
              break;
          ...
          ...
          default:    instrucciones;
      }
```

Es decir, que tiene las siguientes restricciones:

- La expresión cuyo valor hay que analizar debe ser una letra o un número sin decimales. **En caso de haber decimales sólo se pueden usar los `if`.**
- Cada caso se refiere a una igualdad con un valor puntual. **No a rangos ni a igualdades con otras variables.**

Por ejemplo:

```
1  #include <stdio.h>
2  void main() {
3  char estado;
4 printf("Dar inicial de luz de semaforo: ");
5 scanf("%c",&estado);
6   switch(estado )
7     { case 'r':
8         printf("Parar obligatoriamente\n");
9         break;
10     case 'a':
11         printf("Parar si no tienes vehiculo encima\n");
12         break;
13     case 'v':
14         printf("Circular\n");
15         break; }
16      }
```

Fíjate que al final de cada caso va la instrucción `break`.

Plantillas: Condicionales

La instrucción fundamental es

C: IF

Plantilla:

```
if (condicion) {
   instrucciones caso cierto
   }
   else {
   instrucciones caso falso
   }
```

Por ejemplo:

```
1 if (num > horya) {
2 printf("Mayor: %f\n",num);
3 }
4 else {
5 printf("Mayor: %i\n",horya);
6 }
```

ERRORES FRECUENTES: CONDICIONALES

Errores frecuentes

- Confundir = con ==.

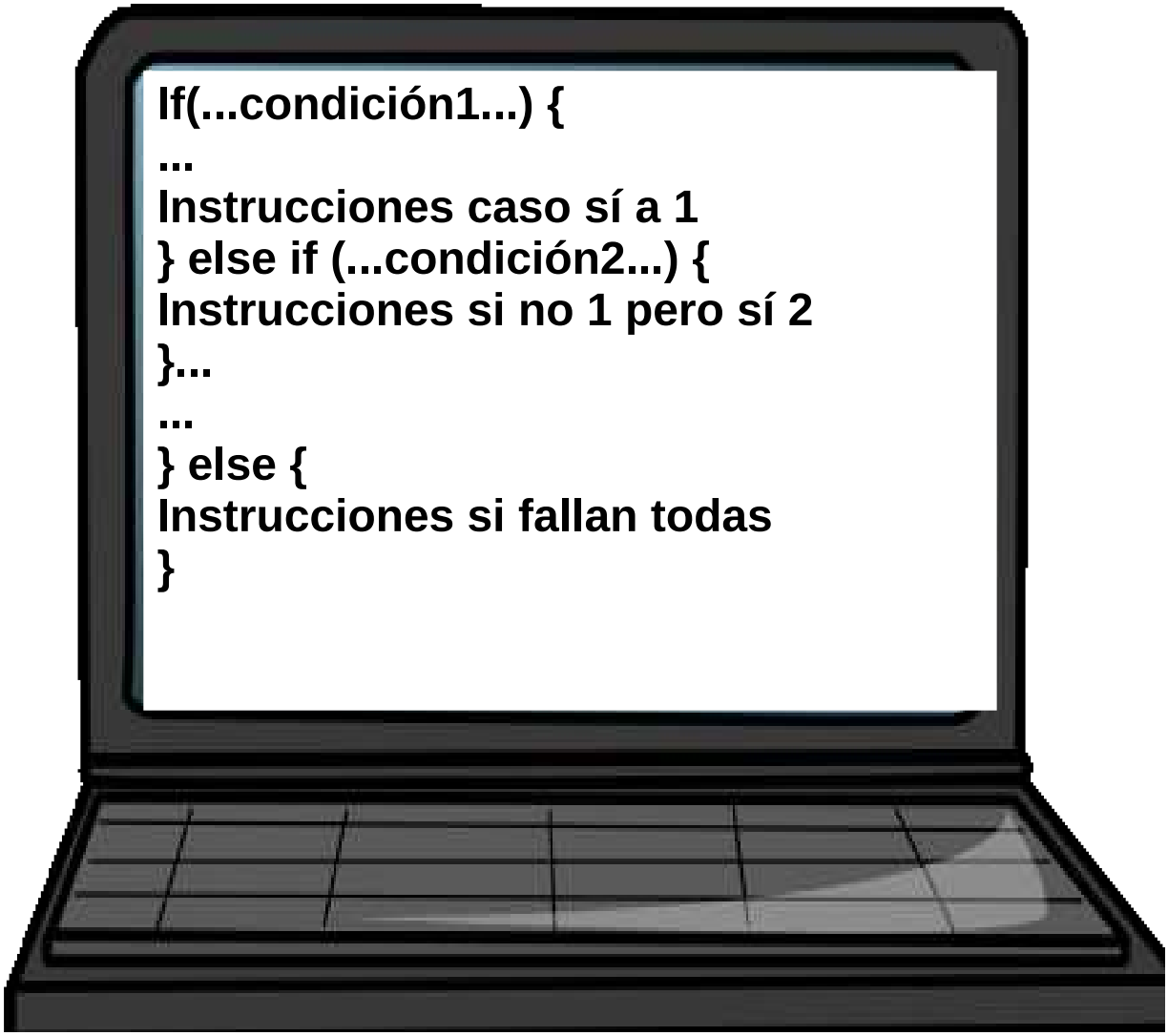
> **PARA MIRAR SI DOS COSAS SON IGUALES EN UN IF HAY QUE PONER ==**

- Es un error poner & en vez de && ó | en vez de ||
- Problemas con las llaves: comprueba siempre que abres y cierras todas.
- Es problemático no poner `break` al final de cada caso en un `switch`.

EJEMPLOS DE INSTRUCCIONES

Aquí se presentan ejemplos de instrucciones de lo que hemos visto.

```
if (a<iniz || a>inder) {    sustituir(&iniz,&inder);    }
   if (caso == 0) {    printf("Opcion no valida\n");    }
   if (a <= b) {    salida = a;    }
```



Cuestiones de autoevaluación

Preguntas

1-En cierto punto hay que hacer ciertas instrucciones si el valor de una variable con decimales x cumple cierta condición. Dar el nombre de la instrucción C necesaria.

2-¿Cómo se pone en C la condición de que x sea mayor o igual que 10? Sólo la expresión, sin el if ni los parntesis y sin espacios.

3-Se quiere leer un número sin decimales, de 1 a 10 y escribir su nombre. Decir qué es más simple de programar if o switch.

4-Si a vale 3 y b vale 2, la condición: (b>a) && (b==2)
es:

a- verdadera
b- falsa
c- está mal escrita
d- faltan datos

5-Se quiere en C un if que compruebe la condición de que x esté entre a y b (sin admitir que sea igual). Es if ((a<x.....<b)) ¿qué hay en medio? No poner espacios.

6-Se tienen que hacer instrucciones distintas según que cierto valor esté entre 0 y 5, entre 5 y 8 o entre 8 y 10. ¿Qué se puede usar if o switch?

7-Se quiere comprobar si el valor de la variable numero (sin decimales) es par, para ello se plantean las siguientes condiciones:
a-(numero%2)==0
b-(numero/2)==(numero/2.)
c-((numero/2.)-(numero/2))==0
Decir las letras de las que valen por el orden presentado, o ninguna si ninguna vale

8-Se quiere comprobar si el valor num2 excede en más de un 30% un límite que está en lim. Se pone:
if(num2>____*lim)
¿Qué irá en el hueco?

9-Se tiene el condicional: if (i%2 == 1)
En caso positivo se pone la instrucción: x++;
En caso negativo (else) se pone: x--;
Antes del condicional i vale 1 y x vale 5. Cuánto valen después. Poner el valor de i y el de x separados por un espacio.

10-Un programa tiene que hacer una operación distinta dependiendo del cuadrante en que esté un ángulo que da el usuario. ¿Cuál es el mínimo número de if que habrá que escribir?

11-Se quiere poner un condicional que verifique si la variable val es 0, será if(.....) ¿Qué va? No poner espacios.

13-Se quiere un condicional que sea afirmativo si la diferencia en valor absoluto entre x1 y x2 es menor que adm. Son con decimales y hay que usarlas por el orden presentado. Dar la cabecera sin espacios.

14-¿En qué casos aparece la instrucción break dentro de un switch?
a-Es obligatoria al principio de cada caso
b-Suele aparecer al final de cada caso
c-Sólo aparece al final del caso default, si es que éste se usa

```
d-No aparece nunca
Marcar la opción elegida, o n si no vale ninguna

15-¿Qué tipos de los siguientes se pueden usar para elegir el caso en un
   switch?
a-int
b-char
c-float
Indicar los posibles juntos, por el orden presentado

16-Un programa lee unas coordenadas en el plano x e y y tiene que
   escribir si está en el primer cuadrante. La condición del if sería
a- (x>0)||(y>0)
b- (x>0)&&(y>0)
c- x,y>0
d- (x>y>0)||(y>x>0)
Dar la opción u opciones válidas juntas, o n si no vale ninguna

18-Se han leído tres valores a, b y c y se quiere hacer cierta acción si
   a es mayor que b y este mayor que c. Posibles condiciones:
a- a>b>c
b- c<b<a
c- (a>b)&&(b>c)
d- (a>b)&&(a>c)
Señalar la(s) que valga(n), juntas y por el orden presentado si hay más
   de una, o n si no hay ninguna

19-Se tiene la instrucción: if((x<0)&&(x>10))
Indicar para cuáles de los siguientes será verdadera:
a) -2
b) 4
c) 17
Darlos juntos por el orden presentado, o n si no hay ninguno

20-Para ver si las variables a y b son iguales se pone: if(a=b). Volver a
   escribir sólo lo que va dentro del paréntesis corregido.

21-Se quiere un programa que lea un número y escriba "Es
   positivo/negativo" según su signo. Alternativas:
a-Un if para ver si es positivo y en el caso afirmativo escribir "Es
   positivo" y en la alternativa else escribir "Es negativo"
b-Escribir "Es " y luego un if para ver si es positivo y en el caso
   afirmativo escribir "positivo" y en la alternativa else escribir
   "negativo"
Dar a,b,ab o n si no vale ninguna

22-Se tiene que leer un número y calcular un resultado con él. Se plantea
```

usar un switch en función del número y meter en los distintos casos el cálculo hecho previamente a mano. ¿Es posible?
a-Nunca
b-Sólo si el número no tiene decimales
c-Sólo si los resultados se repiten mucho
d-Siempre
Indicar la opción elegida, o n si no vale ninguna

Respuestas

1 if

2 x>=10

3 switch

4 b

5)&&(x

6 if

7 abc

8 1.3

9 1 6

10 3

11 val==0

13 if(fabs(x1-x2)<adm)

14 b

15 ab

16 b

18 c

19 n

20 a==b

```
21 ab

22 a
```

Ejemplos resueltos: Condicionales

Decir si un número es par o impar

```
#include <stdio.h>

void main()
   {
   int numero;
   printf("Dar numero: ");

   if (scanf("%i",&numero) == 1
       /* Sólo lo hace si se ha leído 1 número */)
      {
        if (numero % 2 == 0
          /* Es par si el resto de dividir por 2 es 0 */)
           *      {
           *      printf("Es par\n");
           *      }
      }
   }
```

La condición en este caso es si el resto de dividirlo por 2 es 0 o no.

Distinguir sí o no en función de la inicial

```
#include <stdio.h>

void main()
   {
   char inicial;
   printf("Dar inicial (s o n): ");
   scanf("%c",&inicial);
   if (inicial == 's')
      {
      printf("Has dicho s¡\n");
      }
   else
      {
      printf("Has dicho no\n");
      }
   }
```

Sólo se comprueba si es igual a 's', dándose por hecho que si no, es 'n'.

Dar la descripción textual de una intensidad sísmica

```
1  #include <stdio.h>
2  typedef float Richter;
3
4  void main()
5     {
6      Richter inten;
7      printf("Dar intensidad: ");
8      scanf("%g",&inten);
9      if (inten< 5)
10     {
11       printf("Sin problemas\n");
12     }
13     if (inten >= 5 && inten < 5.5)
14     {
15       printf("Algunos problemas\n");
16     }
17     if (inten >= 5.5 && inten < 6.5)
18     {
19       printf("Serio: pueden rajarse paredes\n");
20     }
21     if (inten >= 6.5 && inten < 7.5)
22     {
23       printf("Desastre: pueden caer edificios\n");
24     }
25     if (inten >= 7.5)
26     {
27       printf("Catastrofe: barrios enteros arrasados\n");
28     }
29    }
```

En el caso de que la condición sea estar en un intervalo, lo que se debe poner es "mayor que el extremo inferior y menor que el superior". **No es válido poner** `a<x<b`.

Veamos la construcción paso a paso del programa.

Descripción textual

- Si la intensidad Richter está por debajo de 5, no hay problema
- Si está entre 5 y 5,5 ya puede pasar algo
- Entre 5,5 y 6,5 se pueden romper paredes
- De 6,5 a 7,5 se pueden venir edificios abajo
- Por encima de 7,5 arrasa

Añadir las lecturas y escrituras implicadas

- Pedir la intensidad Richter

- Leerla
- Si la intensidad Richter está por debajo de 5, escribir "no hay problema"
- Si está entre 5 y 5,5 escribir "puede pasar algo"
- Entre 5,5 y 6,5 escribir "se pueden romper paredes"
- De 6,5 a 7,5 escribir "se pueden venir edificios abajo"
- Por encima de 7,5 escribir "arrasa"

Detallar las variables y manifestarlas

Sólo hay una: la intensidad. Puede tener decimales. Vamos a llamarla as: intensidad. Poniéndola explícita en cada instrucción en que salga y separando análisis de cada caso de lo que se hace en él queda:

- Pedir la intensidad Richter
- Leer intensidad
- Si intensidad está por debajo de 5
  - escribir "no hay problema"
- Si está entre 5 y 5,5
  - escribir "puede pasar algo"
- Entre 5,5 y 6,5
  - escribir "se pueden romper paredes"
- De 6,5 a 7,5
  - escribir "se pueden venir edificios abajo"
- Por encima de 7,5
  - escribir "arrasa"

Traducir a C

```
1  printf("la intensidad Richter ");
2  scanf(" %f", &intensidad);
3  if (intensidad < 5) {
4          printf("no hay problema");
5  }
6  if (intensidad>= 5 && intensidad< 5.5 ) {
7          printf("puede pasar algo");
8  }
9  if (intensidad>=5.5 && intensidad<6.5) {
10         printf("se pueden romper paredes");
```

```
11 }
12 if (intensidad>=6,5 && intensidad<7,5 ) {
13         printf("se pueden venir edificios abajo");
14 }
15 if (intensidad>7,5 ) {
16         printf("arrasa");
17 }
```

Añadir partes fijas y declaraciones de variables

Y con eso llegamos ya al programa original.

Dar la luminosidad en función de la potencia

```
1  #include <stdio.h>
2  typedef int Watios, Lumen;
3  /* Las bombillas tienen potencias y luminosidades sin decimales
      */
4
5  void main()
6     {
7     Watios potbomb;
8     Lumen lumbomb;
9     printf("Dar potencia de bombilla: ");
10
11    if (scanf("%i",&potbomb) < 1
12        /* Si no ha leído 1 número  da fallo y termina */)
13       {
14       printf("Error de lectura\n");
15       return  ;
16       }
17    switch (potbomb)
18       {
19       case 15:
20          lumbomb = 125;
21          break;
22       case 25:
23          lumbomb = 215;
24          break;
25       case 40:
26          lumbomb = 500;
27          break;
28       case 60:
29          lumbomb = 880;
30          break;
31       case 75:
32          lumbomb = 1000;
33          break;
```

```
34        case 100:
35           lumbomb = 1675;
36           break;
37        default:
38           lumbomb = -1 /* Sirve para indicar una potencia
                desconocida */;
39        }
40
41     if (lumbomb > 0
42         )
43        {
44        printf("Luminosidad: %i\n", lumbomb);
45        }
46     else
47        {
48        /* Usa un valor negativo
49         para detectar fallo */
50          printf("Potencia desconocida\nLas conocidas son: 15, 25,
                40, 60, 75 y 100\n");
51        }
52     }
```

Se utiliza una instrucción `switch`que comprueba la potencia, dirigiéndose al caso adecuado. Cualquiera que no tenga caso va a `default`

En la figura 0.6 tienes una secuencia animada (si estás viendo el PDF con el visor de Adobe) del programa que distinguía si un número es par o impar. Es para el caso de que el número que le teclees sea 3 Te va marcando la instrucción que va haciendo y a la derecha te saca cómo quedan las variables. La pones en marcha con el botón ▷

```
#include <stdio.h>

void main()
   {
   int numero;
   printf("Dar numero: ");

   if (scanf("%i",&numero) == 1
       /* Sólo lo hace si se ha leído 1 número */)
      {
        if (numero % 2 == 0
          /* Es par si el resto de dividir por 2 es 0 */)
           *      {
           *      printf("Es par\n");
           *      }
      }
   }
```

Variable	Valor
numero	

Figura 0.6: Animación del funcionamiento del programa de mirar si un número es par

¿Cómo se marcan grupos de instrucciones que hay que repetir?

for while do-while

Terminar un ciclo: break

Saltarse una vuelta: continue

¿Qué tipos de ciclos son más frecuentes y cómo se organizan?

C: Ciclos

Llamamos ciclo a un conjunto de instrucciones que se debe repetir unas cuantas veces. Los ciclos surgen en todos los programas que no sean ejemplos sencillos. A partir de ahora, todos los programas tendrán ciclos.

Recursos en el aula virtual de ciclos

Teoría

Aparte de estos apuntes

Ejercicios iniciales de instrucciones concretas

Para comprobar que has interiorizado la teoría
Y recuerda las cuestiones de autoevaluación que tienes aquí en el apartado III

Ejemplos explicados

Con la solución y explicaciones
Y también en estos apuntes, en el apartado 0.8 y en el III

Ejemplos activos

Visualizaciones de la ejecución paso a paso de un programa
Y también en estos apuntes, en la figura 0.12

Ejemplos con solución

La solución sin o con pocas explicaciones
Y también en estos apuntes, en el apartado III

Ejercicios con pistas

Sin memorización de por dónde vas

Ejercicios propuestos

Sin la solución

Controles y exámenes de otros años

Algunos con solución, como ejemplos resueltos, y otros sin ella, como ejercicios propuestos

Cómo escribir ciclos en C

Hay varias instrucciones. Vamos a verlas por orden de popularidad. Estrictamente hablando, con una que se maneje, es posible resolver cualquier caso; pero que se pueda solucionar no quiere decir que sea la forma más cómoda.

Instrucción `for`

Esquema:

```
for (inicialización; condición; incremento) {
       instrucciones;
   }
```

Las partes que aparecen son:

inicialización: lo que se hace antes de empezar las repeticiones

condición: lo que se debe cumplir para seguir repitiendo, es decir, **lo contrario de la condición para terminar**

incremento: lo que se hace al final de cada paso; muchas veces es aumentar un contador, pero puede ser cualquier cosa

instrucciones: las que se deben repetir, una o varias, y pueden ser de cualquier tipo: de cálculo, llamadas a funciones, bloques condicionales, otros ciclos, etc.

Alguna(s) de esas partes puede(n) quedar vacía(s) si no hace falta nada ahí. Si, por el contrario, hacen falta varias cosas en los campos primero o tercero, se ponen separadas por comas.

Ejemplo para escribir los números del 0 al 99:

```
1 #include <stdio.h>
2  void main() {
3   int numero;
4
5          for (numero=0; numero<100; numero++) {
6                  printf("%d\n", numero);
7          }
8 }
```

Instrucción ***while***

Esquema:

```
while (condición) {
      instrucciones;
   }
```

Las partes que aparecen son:

condición: lo que se debe cumplir para seguir repitiendo, es decir, lo contrario de la condición para terminar

instrucciones: el conjunto a repetir, una o varias, y pueden ser de cualquier tipo: de cálculo, llamadas a funciones, bloques condicionales, etc.

Estas partes **no** pueden quedar vacías.

Veamos un ejemplo para repetir bajando un número positivo hasta que se haga 0, sumándolos todos:

```
1  #include <stdio.h>
2  void main()
3  {
4  int n;
5  int suma = 0;
6  /* leer el numero n */
7  printf("n: ");
8  scanf("%i", &n);
9  while (n > 0)
10     {
11          suma = suma + n;
12          n = n - 1; /* equivalente a n-- */
13     }
14 printf("1 + 2 +...+ n = %d\n", suma);
15 }
```

Instrucción `do-while`

El bloque de instrucciones, a diferencia del `while`, se evalúa al menos una vez.
Esquema (atención al ; del final):

```
1 do {
2    instrucciones;
3 } while (condicion);
```

Las partes que aparecen son:

instrucciones: el conjunto a repetir, una o varias, y pueden ser de cualquier tipo: de cálculo, llamadas a funciones, bloques condicionales, etc.

condición: lo que se debe cumplir para seguir repitiendo, es decir, lo contrario de la condición para terminar

Estas partes no pueden quedar vacías.
Veamos un ejemplo para repetir la verificación de si un número es par, siempre que ese número no sea 0:

```
1  #include <stdio.h>
2  void main()
3  {
4  int numero;
5  do
6      {
7      /* se lee el numero */
8      printf("Introduzca un numero: ");
9      scanf("%i", &numero);
10     if ((numero % 2) == 0)
11         printf("El numero %i es par.\n", numero);
12     else
13         printf("El numero %i es impar.\n", numero);
14     }
15 while (numero != 0);
16 }
```

Ciclos con casos especiales

Las **instrucciones de salto** permiten modificar el comportamiento normal de los bucles.

Instrucción `break` Se usa si queremos terminar un ciclo y la comprobación es a mitad del bloque de instrucciones que se repite. En general, sirve para terminar la ejecución de bucles (aparece también en la instrucción `switch`).

Instrucción `continue` Se usa cuando, en ciertos casos, no hay que repetir todas las instrucciones sino sólo una parte, o ninguna, pero sin terminar el ciclo, simplemente pasar a la siguiente vuelta.
Cuando se ejecuta, fuerza un nuevo ciclo del bucle, saltándose cualquier instrucción posterior.
Por ejemplo, para saltarse una parte de las instrucciones a repetir si es que un número es negativo:

```
1  #include <stdio.h>
2  void main () {
3      int n;
4      int positivos = 0;
5
6      do {
7          printf ("\nTeclea un numero (-99 finaliza): ");
8          scanf ("%i", &n);
9          if (n <= 0) continue; //Salta las otras instrucciones
               del ciclo y empieza otra vuelta
10         positivos++;
11     } while (n != -99);
12     printf ("\nHas tecleado %i numeros positivos", positivos);
13 }
```

Salir de varios ciclos anidados con una sola instrucción. La instrucción `break`sale de un ciclo, pero si tenemos un ciclo dentro de otro, sale sólo de uno: el más interior. A veces queremos salir de todos en un solo paso. Aunque eso se puede conseguir con varios condicionales y `break` puede ser más cómoda la instrucción `goto`se usa sólo para salir de varios bucles o ciclos que estén anidados. Por contra, la instrucción `break`sale sólo de uno.

Para usar la instrucción `goto`hay que poner un indicador (una nombre para ese sitio) en la línea siguiente al ciclo del que queramos salir con dos puntos; y donde pidamos el salto de salir poner `goto`y el nombre del indicador, seguido de punto y coma.

Ejemplo:

```
1 for (f=0; f<nfil; f++) {
2         for (c=0; c<ncol; c++) {
3                 if (tabla<0) {
4                         goto fuera;
5                 }
6         }
7 }
8 fuera:
```

ESTILO: CICLOS

Si en un campo de `for`no hace falta hacer nada, sencillamente se deja en blanco, no se pone una variable suelta.

ERRORES FRECUENTES: CICLOS

Errores frecuentes:

- Condición de terminación de ciclos mal puesta. Aparece un ciclo infinito.
- Condición de ciclo que no se cumple nunca y entonces se le salta

- Ojo con poner punto y coma justo detrás de `for` o `while` (pero `do-while` sí lleva al final). El compilador no avisa, así que debemos comprobarlo manualmente.
- Problemas con las llaves (que se nos olvide abrir o cerrar alguna):
  - todas las instrucciones que deben ejecutarse en un ciclo, deben estar dentro de las llaves correspondientes al ciclo
  - las que se deban hacer antes de empezar el ciclo, o después de terminarlo, deben estar fuera de las llaves, antes o después que el ciclo

Ejemplos de instrucciones

Aquí se presentan ejemplos de instrucciones de lo que hemos visto.

```
for ( ;scanf("%i",&valor)==1; acum += valor) {    printf("Eco: %i\n",valor);}
   for(total=0; ; ){    total +=n;    if (total>MAXAC){  break;    }}
   for (base=1; base <= maximo; ) {    base *= factor;}
   for(puntos=SALIDA; adiv!=real; puntos--){ scanf("%i",&adiv);}
   do {    scanf("%i",&primo);} while (primo<=0);
   do {    potencia = pow(potencia,expon);} while(potencia<extremo);
   for (x=0.5; x<10; x+=0.5) {    printf("%f ",x);}
```

Cuestiones de autoevaluación

Preguntas

1-Un programa tiene que contar cuántas veces aparece cierta letra en una palabra que tecleará el usuario. Necesita una variable en que irá contando (eso se llama contador). ¿Qué valor debe darle antes de empezar a mirar la palabra? (contestación numérica)

2-¿Qué tipo de ciclo es apropiado para presentar un menú repetidamente hasta que el usuario pida terminar? Dar el nombre de la instrucción todo junto.

3-Si las variables i,j y k valen 1, 2 y 3 respectivamente y se llega a

```
for(i=j;i<=10;i+=2) {
      if (i==k) {
            k=j;
      }
}
printf("%d",k)
```

¿Qué saldrá en pantalla?

4-Antes de un ciclo do-while las variables j y xx valen respectivamente 5 y 1. Dentro del ciclo van las instrucciones:

```
      {
      xx *= j-1;
      j--;
      }
```

y la condición que va en el while es (j-1)>0
¿Cuál será el valor de xx al terminar el ciclo?

5-Se tiene un ciclo for cuya cabecera es: for(j=1;j<MAX;j++)
Antes del ciclo se sabe que la variable m tiene el valor 50 y que MAX vale 7
Dentro del ciclo están las instrucciones:

```
      {
      l=j;
      m--;
      }
```

¿Cuánto valen l y m al salir del ciclo? Dar los números por ese orden separados por un espacio.

6-Se tiene un ciclo for, dentro del cual están las instrucciones:

```
      {
      suma=0;
      printf("Dame valor: ");
      scanf("%f",&x);
      suma += x;
      }
```

Al ejecutar, el ciclo se repite cinco veces y el usuario da cinco veces el 1. ¿Cuánto valdrá suma al salir del ciclo?

7-Se quiere un ciclo for que repita unas instrucciones dando valores a la variable x desde MIN hasta MAX (ambos incluidos), subiendo de BASE en BASE. Dar la cabecera de este ciclo sin espacios. Hacer el incremento con el operador +=

8-Se quiere un ciclo for que de valores a la variable indice desde 1 hasta k, este último excluido, subiendo de 2 en 2; hacer el incremento con += No poner espacios

9-Un programa lee los primeros términos de dos sucesiones x1 e y1 y va escribiendo los siguientes, usando las fórmulas:
x(n)=x(n-1)+2y(n-1)
y(n)=y(n-1)+2x(n-1)
¿Cuántas variables necesita?

10-Un programa tiene que ir leyendo valores y escribir un mensaje si todos son 0. ¿Hace falta un condicional dentro del ciclo de lectura? ¿Y después del ciclo? Contestar dos palabra sí/no separadas por un espacio.

11-Un programa tiene que repetir unas instrucciones dando valores a la variable iter, desde 0 hasta max (excluido). Dentro del ciclo se va calculando una variable a y el ciclo se terminará anticipadamente si esa variable toma valor negativo. Se tienen dos opciones:
1- Poner la condición en el ciclo: for (iter=0; (iter < max) && (a > 0); iter++)
2- Dejar el ciclo como for (iter=0; iter < max; iter++) y meter dentro un if cuya cabecera sea: if (a < 0). Dentro del if poner la instrucción break;
Decir cuál de estas opciones es válida con su número (ó 12 si ambas valen ó 0 si no vale ninguna)

12-Se lee un valor inicial de una sucesión y1 y hay que ir escribiendo los siguientes términos, usando la fórmula:
y(n)=raiz cubica de y(n-1) ¿Cuál es el mínimo número de variables para almacenar valores de la sucesión? Contestar con un número

13-En un ciclo se van obteniendo unos valores para la variable v y se quiere meter el menor de todos en la variable min. Para ello se pone dentro del ciclo:
if (v < min)
¿Qué instrucción hay que hacer si se cumple la condición anterior?

14-Si se tiene un ciclo dentro de otro y en un punto del ciclo interior se ejecuta la instrucción break, ¿qué pasa?
a-Se termina el ciclo interior
b-Se terminan ambos ciclos
Dar a,b,ab o ninguna

15-Un programa tiene que leer un dinero inicial (variable inicio) en caja (variable caja) y luego sucesivas operaciones de cobros o pagos (variable operacion) y actualizar el dinero en caja. Para ello usa un ciclo. ¿Qué instrucción referida a la variable caja hay que poner antes de entrar en el ciclo? No poner espacios.

16-Un programa tiene que hacer cuatro cosas por turno y luego volver a empezar y así sucesivamente, es decir, hacer A, hacer B, hacer C, hacer D, hacer A, hacer B, hacer C, etc. Para ello se plantea:
a-un ciclo para hacer las repeticiones de A, después un ciclo para hacer las de B, después otro para C y después otro para D
b-un ciclo que cuenta el total de repeticiones con la variable iter y dentro un switch que elija en función del valor iter%4
c-un ciclo que cuente las repeticiones de A, dentro otro que cuente las de B, dentro de éste otro que cuente las de C y dentro de éste otro que cuente las de D
d-un ciclo que cuenta el total de repeticiones y dentro un switch que

elige en función de una variable toca, al ejecutar cada caso se le da a toca el valor del siguiente caso
e-un ciclo que cuenta las repeticiones avanzando de cuatro en cuatro y dentro hacer A, B, C y D
Dar las opciones válidas por el orden presentado, sin espacios entre ellas si son varias, o ninguna si ninguna vale

17-¿Qué tipo de ciclo es apropiado para pedir y leer un dato y volverlo a pedir si lo que se mete no vale? Dar el nombre de la instrucción todo junto.

18-Un ciclo tiene la cabecera: for(iter=0; iter < lim; iter++)
Dentro de él se comprueba cierta condición y se corta el ciclo si esa condición se cumple, aunque no se haya llegado a lim. Si después del ciclo hay que distinguir si se ha salido por haber terminado las iteraciones o por haberse cumplido la condición ¿qué se puede poner?
if (iter.....)
Se quiere que sea cierto en el caso de que se haya salido por la condición. Terminarlo sin espacios.

19-En un programa están las dos siguientes instrucciones
```
#define VERDADERO (1==1)
#define FALSO (1==0)
```
Queremos que un ciclo while (...) no corte las repeticiones nunca. ¿Se puede usar alguna de las expresiones anteriores entre los paréntesis del while? Contestar VERDADERO, FALSO, ambas o ninguna.

20-Queremos que un ciclo se pare cuando el contador cont llegue a max y también si la variable error es más pequeña que admi. Alternativas:
a- (cont<max) || (error>admi)
b- (cont<max) && (error>admi)
c- (cont>=max) || (error>=admi)
d- (cont>=max) && (error>=admi)
Dar las condiciones válidas sin separación y por el orden presentado, o ninguna si ninguna vale

21-Decir el mínimo posible de veces que se ejecuta un ciclo de tipo for, uno de tipo while y otro de tipo do-while (números separados por un espacio y por este orden)

22-Si numeramos los campos de una instrucción for como 1, 2 y 3, es decir,
for (..1..;...2...;...3...)
Dar los números de los que se pueden dejar en blanco, por el orden presentado y separados con un espacio.

23-Dar, seguidas y por el orden presentado, las frases verdaderas:
a) se puede meter un ciclo dentro de un condicional if

```
b) se puede meter un ciclo dentro de un caso de un condicional switch
c) se puede meter un condicional dentro de un ciclo for
d) se puede meter un condicional dentro de un ciclo do-while

24-Decir qué instrucción permite hacer más ciclos, es decir, ciclos que
   con otras no se puede hacer el programa
a-for
b-while
c-dowhile
Dar la letra o letras de las que permiten más, juntas y por el orden
   presentado

25-Se tiene un ciclo con la cabecera: for(i=1; i<10; i*=2)
Dar separados por espacios todos los valores que tomará i, excluyendo
   cuando se salga del ciclo

26-Para hacer un programa que lea números y los sume hasta que el usuario
   introduzca una letra, se plantea el siguiente ciclo:
   for(suma=0;scanf("%f",&numero)!=____;suma+=numero)
Dentro del ciclo no hay ninguna instrucción. ¿Qué va en el hueco?

27-Un programa usa un ciclo para ir variando unos niveles de radiacion en
   la variable radia. La cabecera del ciclo es:
   for(radia=inicio;radia>min;radia/2)
Si todas las variables son sin decimales, inicio vale 100 y min vale 20,
   dar todos los valores que va a tener radia dentro del ciclo.

28-La media geométrica de una serie de números es su producto elevado al
   inverso de la cantidad de ellos. Para resolver esto se usa un ciclo en
   que a una variable acumulador se le va multiplicando por cada número y
   esa variable es la que se eleva al inverso de la cantidad usando la
   función pow. El acumulador debe empezar valiendo 1. ¿En qué posición
   se pone a 1 y en qué posición va la llamada a pow?
a-Antes de meterse en el ciclo
b-Dentro del ciclo
c-Después de salir del ciclo
Contestar con las dos letras juntas por el orden que se ha preguntado?

29-La cabecera de un ciclo es for(con=1;con<=10;con+=2)
¿Cuántas veces se ejecutarán las instrucciones que están dentro del ciclo?

30-Se quiere un ciclo en que la variable cont vaya pasando de 1 hasta n
   incluidos, pero saltándose el valor exc. Estrategias alternativas
a-Usar un ciclo for poniendo como condición (cont <= n)&&(cont != excl)
b-Usar un ciclo for con la condición (cont <= n) y dentro poner lo
   primero if (cont == excl) y si se cumple poner la instrucción continue;
c-Usar un ciclo for con la condición (cont <= n) y dentro poner lo
```

primero if (cont == excl) y si se cumple poner la instrucción break;
d-Usar un ciclo for con la condición (cont <= n) y dentro poner lo primero if (cont != excl) y si se cumple poner las instrucciónes que haya que hacer, la alternativa else dejarla vacía
Dar las opciones válidas seguidas, o n si no vale ninguna

31-Se quieren obtener todas las parejas posibles de números de 1 a n incluidos. Estrategias alternativas:
a) Un ciclo for(i=1,j=1;i<=n,j<=n;i++,j++) y dentro escribir i y j
b) Un ciclo for(i=1;i<=n;i++) y dentro otro ciclo for(j=1;j<=n;j++) y dentro de éste escribir i y j
c) Un ciclo for(i=1;i<=n;i++) y dentro una instrucción j=1; otra para escribir i y j y otra j++;
Dar las opciones válidas, o n si no vale ninguna

32-Se quieren escribir numval valores crecientes de 1 en 1 empezando en num1 y luego decrecientes de 1 en 1 hasta completar entre los crecientes y los decrecientes un total de totnum valores. Para ello se van a usar ciclos. Estrategias alternativas para el control de repeticiones:
a-Usar la variable a escribir, calculando sus límites para que cumplan lo pedido
b-Usar otra variable que no es el valor a escribir, sino la cuenta de los que se van escribiendo
Dar las opciones válidas a, b, ab o b si ninguna vale

33-Se quiere un ciclo que repetidamente pida un valor y haga un cálculo con él si el valor proporcionado sea negativo, en cuyo caso no hará más repeticiones. Estrategias alternativas:
a-Poner la lectura del valor y luego for(;valor>0;) y dentro (hacer el cálculo y pedir el siguiente valor)
b-Poner la lectura del valor y luego do-while(valor>0) y dentro (hacer el cálculo y pedir el siguiente valor)
c-Poner la lectura del valor y luego while(valor>0) y dentro (hacer el cálculo y pedir el siguiente valor)
Dar las opciones válidas juntas, o n si no vale ninguna

34-Se quiere ir obteniendo los números que cumplen cierta condición, hasta haber juntado una cierta cantidad que lo cumple. ¿Cuántas variables hacen falta?

35-Se quiere sumar los números positivos de una serie que va a introducir el usuario. Finalmente habrá que escribir esa suma. Para ello se usa un ciclo que cuenta hasta la longitud de la serie. Dentro se pone la lectura de número y un if que comprueba si el número es positivo. ¿Dónde hay que poner la suma a 0, donde hay que sumarle el número y dónde hay que poner la escritura de la suma? Elegir para cada una de

```
   entre las siguientes posiciones:
a-Antes de entrar en el ciclo
b-Dentro del ciclo, antes de la lectura
c-Dentro del ciclo, entre la lectura y el if
d-Dentro del if
e-Dentro del ciclo, después del if
f-Después del ciclo
Dar las tres letras seguidas
```

Respuestas

1 0

2 dowhile

3 3

4 24

5 6 44

6 1

7 for(x=MIN;x<=MAX;x+=BASE)

8 for(indice=1;indice<k;indice+=2)

9 4

10 sí sí

11 12

12 1

13 min=v;

14 a

15 caja=inicio;

16 bde

17 dowhile

18 <lim

```
19 VERDADERO

20 b

21 0 0 1

22 1 2 3

23 abcd

24 abc

25 1 2 4 8

26 0

27 100 50 25

28 ac

29 5

30 bd

31 b

32 ab

33 ac

34 2

35 adf
```

¿Qué es una función?

¿Cómo divido un pograma único en funciones?

¿Qué son los argumentos?

¿Cómo sé cuáles son de entrada, cuáles de salida y cuáles locales?

¿Cómo se indican y se trabaja con ellos?

C: Funciones

Una función es un bloque de instrucciones que realiza una determinada tarea. Todo programa C consta de una o más funciones, estando siempre presente la función `main()` que es por donde comienza la ejecución el programa. A partir de ahora será obligatorio que nuestros programas tengan alguna función además de `main`.

La tarea que hace una función puede ser parte del flujo de operación normal del programa o ser algo que hay que hacer en alguna circunstancia automática. Vamos a tratar primero las que se ejecutan cuando nosotros programemos que lo hagan.

No hay reglas fijas de cómo dividir un programa en funciones, ni hay solución única. Cualquier reparto del trabajo más o menos equitativo vale, pero debemos recordar que los bloques (condicionales, ciclos) tienen que empezar y terminar en la misma función; no pueden comenzar en una y terminar en otra.

Desde una función se pasa a otra mediante una *llamada* que deberemos poner en el sitio que corresponda.

Una función también puede ejecutarse sin que se la llame, si está controlada por cierta señal que se active (por ejemplo, un temporizador). Esas las comentaremos después. Ahora estamos hablando de las que se ejecutan mediante llamadas puestas por nosotros en el programa.

Las variables de cada función son independientes, aunque estén enlazadas por la llamada. Esto quiere decir que en una función son desconocidas las variables de otras funciones. Para pasar valores de una a otra se usan los *argumentos*. Cada función tiene los suyos. Pueden ser de entrada, salida o ambas cosas. Los primeros son variables que toman el valor de otras ubicadas en la función desde donde se ha hecho la llamada. Los de salida dan valores a otras variables de la función que ha llamado a esta.

Para evitarnos escribir unas líneas extras con prototipos de las funciones, pondremos cada función antes que cualquier llamada a ella, por lo que `main` quedará la última. Pero ese es el orden en que quedarán cuando hayamos acabado el programa; normalmente empezaremos escribiendo `main` y luego nos situaremos más arriba, para poner las funciones encima.

La ejecución empezará por `main` y pasará a otra función cuando encuentre una llamada (o si se dispara una señal, lo que veremos al final). Vamos a empezar por ver las llamadas y luego vemos cómo se define una función.

Valores de retorno

Además de los argumentos de salida, una función puede devolver un valor (solo uno) en su llamada, mediante una instrucción especial. Usarlo o no, es una cuestión más que nada de preferencias, aunque un caso puede ser más legible que otro, según sea el programa. Veremos los casos por el siguiente orden:

funciones sin retorno si devuelven algo es por argumentos de salida

funciones con retorno envían un valor adicional en la llamada, aparte posibles argumentos de salida

Llamadas a funciones sin retorno

Para llamar a una función en algún sitio del programa, se especifica su nombre y la lista de argumentos; así para llamar a una función que se llamase `potencia`, un ejemplo de utilización sería:

```
1 #include <stdio.h>
2
3 void main() {
4  int y; float x, pot;
5  printf("Introduzca dos numeros: ");
6  scanf("%f %d", &x, &y);
7  potencia(x,y, &pot);            /* llamada a la funcion */
8  printf("La potencia es %f\n", pot);
9 }
```

Los parámetros o argumentos de una función pueden ser de entrada (es el caso general y técnicamente se llama *paso por valor*) o de salida (técnicamente se llama *paso por referencia*). Si son parámetros de entrada (por valor) se copia el valor del argumento en la función (técnicamente *parámetro formal*) y dentro de ella opera internamente con esta copia, no con el valor original. En cambio, cuando son de salida (por referencia) cualquier cambio que produzca la función sobre esa variable, tiene efecto en su valor al retorno (técnicamente se pasa un *puntero* al valor).

Cuando queremos que los argumentos de una función sean de salida, si es que se trata de valores individuales (no listas ni tablas, que veremos más adelante y que no utilizan esto) usaremos en la llamada el operador dirección, &. También se llama "de referencia". Es el caso del tercer argumento en el ejemplo anterior.

Definición de funciones sin retorno

En C, para escribir una función se debe definirla empezando por una línea inicial, de cabecera, siguiendo la sintaxis:

```
/* definición */
   void nombre_func (tipo1 arg1, ..., tipoN argN) {

      /* CUERPO DE LA FUNCION */
   }
```

`void` indica que es sin retorno. Cuando alguno de los argumentos es de salida, hay que usar con él el operador de indirección,*. Lo usaremos en la cabecera y en todas partes donde aparezca esa variable dentro de la función. Un ejemplo de declaración y definición acorde con el de llamada visto anteriormente sería:

```
1 void  potencia (float x, int y, float *prod) /* definicion */
2 {
3 int i;
4 *prod = 1;
```

```
5  for (i = 0; i < y; i++) {
6      *prod = (*prod) * x;
7  }
8  }
```

Otro ejemplo (obsérvese como al final ha quedado la función antes que `main`):

```
1
2  #include <stdio.h>
3
4  void swap(int *a, int *b) {
5  /*Funcion que intercambia el valor de dos variables. Paso de
6  argumentos por referencia (para que puedan salir).*/
7   int temp;
8   temp = *b;
9   *b = *a;
10  *a = temp;
11 }
12
13 void main() {
14 int x = 2;
15 int y = 5;
16 printf("Antes x = %i, y = %i\n", x, y);
17 swap(&x, &y);
18 printf("Despues x = %i, y = %i\n", x, y);
19 }
```

Llamadas a funciones con retorno

Es igual, pero a la izquierda hay que poner la asignación de ese valor que se devuelve a alguna variable; así, si hiciésemos de esta forma el caso de la función `potencia`, vista anteriormente, quedaría:

```
1          #include <stdio.h>
2
3          void main() {
4                  int y; float x, pot;
5                  printf("Introduzca dos numeros: ");
6                  scanf("%f %d", &x, &y);
7                  pot=potencia(x,y);            /* el resultado no
                      va por argumento de salida, sino por valor de
                       retorno */
8                  printf("La potencia es %f\n", pot);
9          }
```

Si solo hay un valor de salida y era argumento de solo salida, ya no hace falta argumento de ese tipo, sino que se asigna a una variable. Cuando un argumento es de entrada y salida, si esa salida va a valor de retorno, entonces el argumento pasaría a ser solo de entrada.

Definición de funciones con retorno

En este caso la cabecera es:

```
/* definición */
   tipo nombre_func (tipo1 arg1, ..., tipoN argN) {

        /* CUERPO DE LA FUNCION */
```

`tipo` es el que corresponda al valor de retorno (`int`, una unidad definida, `float`, ...). La declaración de los argumentos es igual.

Además, donde vaya a acabar la función hay que poner la instrucción donde se indica qué valor es el que vuelve por retorno. Para ello se usa la instrucción `return` de la forma `return valor;`

`valor` puede ser una variable, una expresión de cálculo, etc.

Traduciendo el ejemplo visto anteriormente a esta casuística, quedaría:

```
1  float  potencia (float x, int y) /* la salida por
       retorno va a ser float */
2  {
3          int i;
4          float prod = 1;
5          for (i = 0; i < y; i++) {
6                  prod = prod * x;
7          }
8          return prod; // el valor que se devuelve por
               retorno es el de prod
9  }
```

Funciones activadas por señales automáticas

Vamos a comentar ahora el caso de que queramos que una función entre automáticamente cuando suceda cierta señal, sea lo que sea lo que esté haciendo el programa en ese momento. Las señal que vamos a considerar es enviada desde teclado (Ctrl+C)

Vamos a ver paso a paso cómo tenemos que preparar estas funciones:

1. Añade `#include <signal.h>` al principio del programa.

2. Prepara una función para responder a la señal. Recibirá un solo argumento int que es el código de la señal. Pon allí las instrucciones que sean. Si quieres que responda más de una vez tienes que ponerle al final la instrucción de activación; se comenta a renglón seguido.

   En esa función, no escribas ni leas en los mismos sitios donde lo haga el resto del programa, porque podrías interferir desastrosamente con una operación que esté a medias.

3. Enlaza la señal a la función, poniendo `signal(SIGINT, nombredetufuncion);`

   Esto mismo tendrás que ponerlo también al final de la función si quieres que funcione más de una vez.

Ejemplo

```
1 #include <stdio.h>
2 #include <locale.h>
3 #include <signal.h>
4 #include <stdlib.h>
```

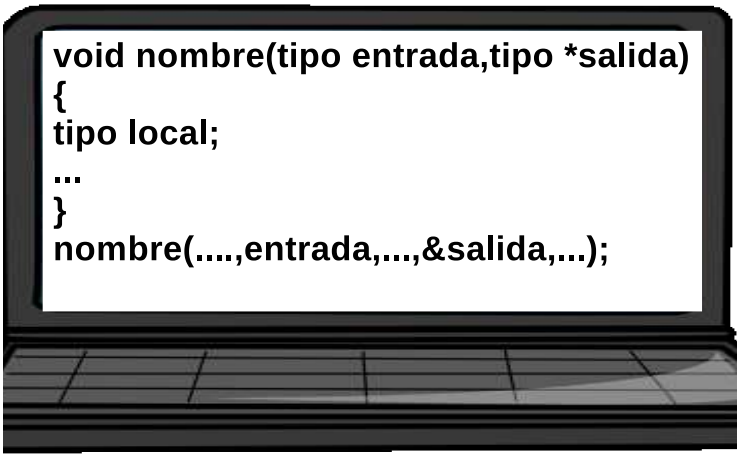
```
5
6  void terminar(int s) {
7  /*Funcion que se activa con Control-C.
8  Saca un mensaje y termina el programa*/
9   printf("Sus deseos son órdenes, alteza. Me paro.\n");
10  exit(EXIT_SUCCESS);
11  //No se reactiva porque sólo va a funcionar una vez
12 }
13
14 void main() {
15
16 setlocale(LC_ALL,"");
17 signal(SIGINT,terminar);
18 for ( ; ; ) {
19 //Ciclo que no hace nada y no termina
20 }
21 }
```

Estilo: Funciones

- Se denominarán con verbos, relacionados con su propósito
- Sus nombres irán en minúsculas.
- Separar tareas diferentes en funciones distintas
- Repartir el programa en funciones. No se considera reparto si una función lo único que hace es llamar a otra, ni si la llamada es sin argumentos. Las funciones complejas (demasiadas líneas, muchas estructuras condicionales o de ciclo, o mucho anidamiento de bloques dentro de otros) deben dividirse en varias. La complejidad resultante debe ser intermedia.
- Las funciones llevarán un comentario indicando sus objetivos, a qué resultados pretenden llegar en función de qué datos, salvo que sea obvio

Errores frecuentes: Funciones

- Grave error: poner ; justo detrás de la cabecera de una función. El compilador no avisa, así que debemos comprobarlo manualmente.
- Al revés, también es error no poner ; en instrucciones de llamada a función
- Problemas con las llaves (no abrir o cerrar alguna llave correspondiente a principio o fin de función).
- No puede haber instrucciones ejecutables fuera de ninguna función (todas deben estar dentro de alguna)
- Cuidado con confundir el orden o tipo en los argumentos de una función entre llamada y definición. El compilador no se fija en el nombre, sino en el orden.



Ejemplos de instrucciones

Aquí se presentan ejemplos de instrucciones de lo que hemos visto.

```
leer(&constante,&distancia);
   if (opcion != 's') {    menu();} else {    printf("Vuelva a repetir
      opcion\n");}
   void operar(float op1, float op2, float *suma, float *resta) {...}
   for(iter=0; funap!=0; iter++){      aproximar(x,&funap);}
   void sumar(int cuantos, Euros cuenta) {...}
   cambiar(inicio,fin,serie);
   escribir(carga);
   if (a > 0 && (b*b)>=c) {    resol2(a,b,c,&s1,&s2);} else {
      resol1(b,c,&s1);}
   void escselec(int nreal, int hojas[HOJPLAN]) {...}
```

Cuestiones de autoevaluación

Preguntas

1-Dar el número mínimo de resultados que puede dar una función al programa que la llama.

2-Hace falta una función que reciba un ángulo en grados sexagesimales y devuelva el seno del ángulo doble que él. Al principio del programa se tiene:
#define FACTOR (6.2832/180)
El argumento con el ángulo dato se llama a. Dentro de la función sólo hay dos líneas y la segunda es:
*s2=sin(2*r);
Dar la primera sin espacios y de forma que el primer operando sea a.

3-Dar el número mínimo de argumentos que puede tener una función en C.

4-La cabecera de una función es: void calcular(float x, float a, int *j)
El resultado es de tipo:
a: número con decimales
b: número sin decimales
c: letra
d: no tiene resultado
Dar sólo la(s) letra(s) correspondiente(s), juntas y por orden si hay más de una, o n si ninguna vale

5-La llamada a una función (correcta) es:
compara(gamma1,gamma2,&x);
Todas las variables son con decimales. Dar la cabecera de la función, conservando los nombres de las variables y sólo con los espacios imprescindibles (no detrás de las comas).

6-La cabecera de una función es:
void comer(float x, float y, float *z)
Dar una instrucción de llamada conservando los nombres de las variables y sin espacios.

7-¿Son posibles estas cabeceras?
a-void probar()
b-void probar(int a)
c-void probar(int a, float *b)
d-void probar(int *a, float *b)
Contestar con las letras de las posibles juntas por orden alfabético o n, si ninguna es posible

8-Dentro de una función (distinta de main) ¿se puede llamar a otra?
a-No, a ninguna
b-Sí, a cualquiera
c-Sí, salvo a main
d-Sí, salvo a sí misma
Dar la opción correcta, o ninguna si ninguna vale

```
9-Los nombres de los argumentos en una función y de las variables
   correspondientes en la función que la llama...
a-deben coincidir
b-pueden coincidir o no
c-no deben coincidir
Dar la letra correcta.

10-Pueden llamarse igual variables de distintas funciones
a-Nunca
b-Sólo si son argumentos
c-Sólo si no son argumentos
d-Siempre es posible
e-Siempre es obligatorio
Dar las letras válidas seguidas por el orden presentado

11-¿Cuántas llamadas a una misma función pueden aparecer como máximo en
   un programa?
a-Ninguna
b-Una
c-Una desde cada una de las otras funciones
d-Las que se quiera
Dar la letra correcta

12-¿Cuál es el mínimo número de llamadas que puede haber a una función?
a-Ninguna
b-Una
c-Una, salvo main que puede no haber ninguna
d-Una desde cada una de las otras funciones

13-Un programa llama tres veces a una función con las siguientes
   instrucciones:
calcular(a,&ra);
calcular(b,&rb);
calcular(c,&rc);
La cabecera de la función es: void calcular(int x, int *y)
Dentro de la función sólo hay una instrucción: *y = x*x;
Antes de las llamadas a, b y c valen 3, 4 y 5. ¿Qué valdrán después de
   las tres llamadas ra, rb y rc?
Dar los tres números separados por espacios.

14-En un programa una función lee un valor, otra función obtiene dos
   resultados a partir de él y otra función los escribe en pantalla. Son
   llamadas sucesivamente desde main. ¿Cuántos argumentos resultado tiene
   cada una?

15-En un programa una función lee un valor, otra función obtiene dos
   resultados a partir de él y otra función los escribe en pantalla. Son
```

llamadas sucesivamente desde main. ¿Cuántos argumentos dato tiene cada una?

16-La cabecera de una función es: void operar(float x, float y, float *temp)
Dentro están las instrucciones:
*temp=x;
x=y;
y=(*temp);
Se llama a esta función con la instrucción: operar(a,b,&z)
Antes de la llamada valen 1, 2 y 3 respectivamente. Dar lo que valen después, por ese orden y separados por espacio.

17-Una función se encarga sólo de escribir el título de un programa, que es su nombre con mayúsculas. ¿Cuántos datos y cuántos resultados necesita? Dar los dos números separados por espacio.

18-Señalar la(s) verdadera(s):
a-En lo que queda de curso habrá que usar funciones
b-Las funciones sólo se usan en este tema
c-Las funciones sólo se usan en este tema y en el último, cuando se den tablas
d-Las funciones se dan informativamente, pero no se piden y no se valoran

19-¿Se puede reutilizar el nombre de una función?
a-Sí, si interesa para otra cosa
b-Se puede para una variable, pero no para otra función
c-Se puede para otra función, pero no para variables
d-No se puede
Dar la letra correcta, o n si no vale ninguna

20-¿Cuál es el número máximo de funciones que puede tener un programa?
a- 1
b- 2
c- 3
d- 4
Dar la letra correcta, o n si no vale ninguna

21-¿Cuál es el número mínimo de funciones que puede tener un programa?
a- 0
b- 1
c- 2, si es modular
d- 3
Dar la(s) letra(s) correcta(s), juntas y por este orden si hay más de una, o n si no es ninguna

22-Decir cuáles de los siguientes nombres de funciones valen

a) 4saumare
b) sumar_1
c) ordenar-inverso
d) multiplicar12
Dar la(s) letra(s) correcta(s), juntas y por este orden si hay más de una, o n si no es ninguna

Respuestas

1 0

2 r=a*FACTOR;

3 0

4 b

5 void compara(float gamma1,float gamma2,float *x)

6 comer(x,y,&z);

7 abcd

8 b

9 b

10 d

11 d

12 c

13 9 16 25

14 1 2 0

15 0 1 2

16 1 2 1

17 0 0

18 a

19 d

```
20 n

21 bc

22 bd
```

Ejemplos resueltos: Programas muy sencillos

Pasar de pulgadas a centímetros

```
1  #include <stdio.h>
2  /* Una pulgada son 2.54 centmetros */
3  #define FACTOR 2.54
4  typedef float Pulgadas, Centimetros;
5
6  void convierte(Pulgadas puldato, Centimetros *resultado)
7     {
8     *resultado=(Centimetros) (FACTOR * puldato);
9     }
10
11 void main()
12    {
13    Pulgadas puldato;
14    Centimetros centires;
15    printf("Dar pulgadas: ");
16    scanf("%g",&puldato);
17    convierte(puldato,&centires);
18    printf("En centmetros es: %g\n", centires);
19    }
```

La función tiene un argumento de entrada, las pulgadas, y otro de salida, los centímetros.

Pasar de Kibibytes a Mebibytes

```
1  #include <stdio.h>
2  /* Un MiByte son 1024 KiByte */
3  #define FACTOR 1024
4  typedef int Kibyte, Mibyte;
5  /* Se van a trabajar sin decimales */
6
7  void convierte(Kibyte kbdato,Mibyte *resultado)
8     {
9     *resultado=(Mibyte) (kbdato / FACTOR);
10    /* Se indica específicamente la conversión de unidades */
11    }
12
```

```
13 void main()
14    {
15    Kibyte dato1, dato2, dato3;
16    Mibyte resul1, resul2, resul3;
17    printf("Dar tres valores en KiBytes: ");
18    scanf("%i %i %i",&dato1, &dato2, &dato3);
19    convierte(dato1,&resul1);
20    convierte(dato2,&resul2);
21    convierte(dato3,&resul3);
22    printf("En MiBytes son: %i %i %i\n", resul1, resul2, resul3);
23    }
```

En este caso la función se llama tres veces con distintos datos. El nombre del argumento no coincide con ninguno de las llamadas. En cada ocasión es sustituido por lo que se le pasa. Conclusión: **no es necesario hacer varias funciones para un mismo cálculo con distintos datos, basta llamar a una única función variando los argumentos en las llamadas**.

Radios de circunferencias inscrita y circunscrita a un triángulo

```
1  /* 04/04/2006 13:34:26 */
2  #include <stdio.h>
3  #include <math.h> /* Para la función sqrt, que hace la raíz
      cuadrada */
4  /* El radio de la inscrita es el lado dividido por la raíz de 12
5  y el de la circunscrita es dividido por raíz de 3 */
6  #define DIVINS sqrt(12.)
7  #define DIVCIR sqrt(3.)
8  typedef float Metros;
9
10 void calins(Metros lado, Metros *radins)
11    {
12    *radins=lado/DIVINS;
13    }
14
15 void calcir(Metros lado, Metros *radcir)
16    {
17    *radcir=lado/DIVCIR;
18    }
19
20 void main()
21    {
22    Metros lado, rins, rcir;
23    printf("Dar lado: ");
24    scanf("%g",&lado);
25    calins(lado,&rins);
26    calcir(lado,&rcir);
```

```
27    printf("El radio de la inscrita es: %g y el de la
         circunscrita es: %g\n",
28     rins, rcir);
29    }
```

Calcular peso de chapa circular

```
1  /* 04/04/2006 13:37:10 */
2  #include <stdio.h>
3  #define PI 3.1416
4  typedef float Metros, Metros2, Metros3, Kpm3, Kilos;
5  /* El tipo Kpm3 lo usaremos para Kilos por metro cúbico */
6  /*~TypeCombination
7  Metros * Metros -> Metros2 ,
8  Metros2 * Metros -> Metros3 ,
9  Metros3 * Kpm3 -> Kilos */
10
11 void calarea(Metros radio, Metros2 *area)
12 /* Calcula área de círculo */
13    {
14    *area = PI * radio * radio;
15    }
16
17 void calvol(Metros radio, Metros espesor, Metros3 *volumen)
18 /* Calcula volumen usando función para área y
19  multiplicando por espesor */
20    {
21    Metros2 area;
22    calarea(radio,&area);
23    *volumen=area * espesor;
24    }
25
26 void calpeso(Metros radio, Metros espesor, Kpm3 densidad, Kilos
      *peso)
27 /* Calcula peso usando función para volumen y
28 multiplicando por densidad */
29    {
30    Metros3 volumen;
31    calvol(radio, espesor,&volumen);
32    *peso = volumen * densidad;
33    }
34
35 void main()
36    {
37    Metros radio, espesor;
38    Kpm3 densidad;
39    Kilos peso;
```

```
40    printf("Dar radio, espesor y densidad: ");
41    scanf("%g %g %g",&radio, &espesor, &densidad);
42    calpeso(radio,espesor, densidad,&peso);
43    printf("El peso es: %g\n", peso);
44    }
```

En este caso hay varias funciones y se llaman unas a otras. Como se ve, no siempre las funciones se tienen que llamar desde `main`. Se llaman cuando haga falta.

Programa que opera según se le pida

En la figura 0.7 tienes explicado un programa que hace una operación aritmética que le pida el usuario, pudiendo elegir entres las cuatro elementales. No está en C directamente, sino en diagrama de cajas, pero es muy directo pasarlo a C.

Plantillas: Ciclos

Repetir un cierto número de veces

Plantilla:

```
for(contador = 1; contador <= numeroveces; contador++) {
   instrucciones de proceso a repetir
    }
```

Ejemplo:

```
1 for(contador = 1; contador <= pedido; contador++) {
2 printf("%i %i\n",contador,contador*contador);
3 }
```

Repetir hasta cierta condición

Si la condición se controla fácilmente y es al principio de cada vuelta (la condición de seguir es opuesta a la de terminar), la plantilla puede ser:

```
for(acción inicial; condición de seguir; cambio en cada vuelta) {
   instrucciones de proceso
   }
```

Ejemplo

```
1 for( ; suma <= limite; ) {
2 scanf("%f",&valor);
3 suma+=valor;
4 }
```

Si la condición es complicada o no se hace al principio de cada vuelta, sino en mitad, la plantilla podría ser:

```
for(acción inicial;  ; cambio en cada vuelta) {
   instrucciones de proceso
   if (condicion de salir) break;
   otras instrucciones de proceso
   }
```

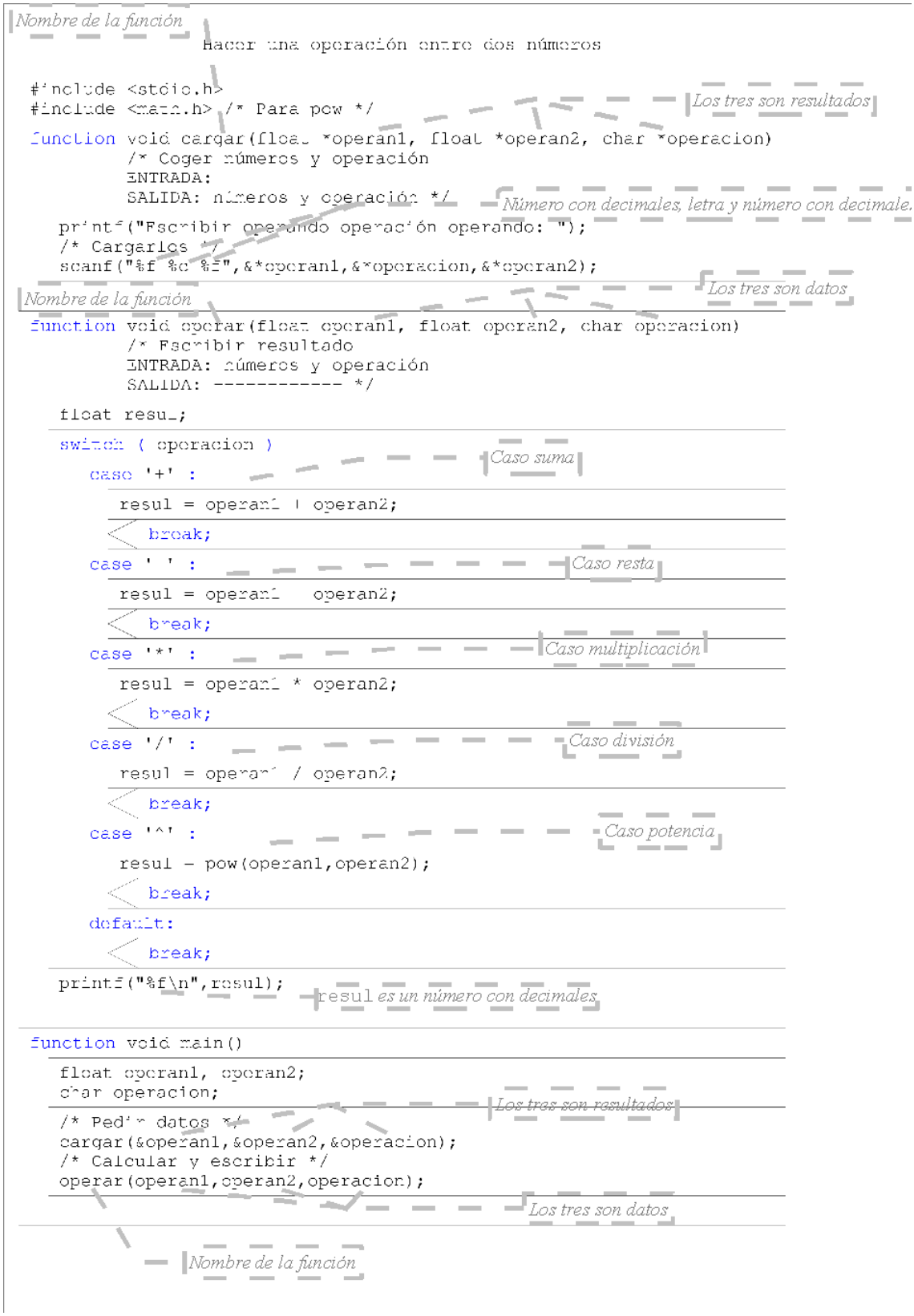


Figura 0.7: Programa que hace una operación u otra según se le pida

Ejemplo:

```
1 for( ; ; ) {
2 scanf("%f",&valor);
3 if (valor < 0) break;
4 suma+=valor;
5 }
```

Programa que calcula nivel de radiación

En las figuras 0.8, 0.9 y 0.10 se presenta respectivamente el enunciado, explicación y solución del programa. Es del tipo de plantilla antedicha. Está tomado del cuaderno de ejercicios del profesor Pedro Corcuera.

> **Programa que calcula la radiación de una sustancia cada tres días indicando además si el nivel es seguro. La radiación inicial es de 150 mr (milirems) y la radiación decrece la mitad cada tres días. La radiación se considera segura por debajo de 0.466 mr. y el proceso debe parar antes de alcanzar una radiación de 1/10 del nivel seguro.**

Figura 0.8: Enunciado de programa para calcular el nivel de radiación hasta llegar a valor seguro

Ir dando valores calculados

Plantilla:

```
for(valor inicial; condicion de seguir; cambio de valor) {
   instrucciones de proceso
   }
```

Ejemplo:

```
1 for(x = maximo; x >= minimo; x /= 2) {
2 printf("%f %f\n",x,log10(x));
3 }
```

Cuenta de elementos leídos

Se trata de un recorrido leyendo (ver plantilla anterior) en el que se van contando los valores que cumplen cierta condición (o todos). La variable en que se cuenta tiene que ponerse a 0 antes del ciclo.

Plantilla:

```
cuenta=0;
   for(accion inicial; condicion de seguir; cambios) {
   scanf valor
   instrucciones de proceso
   sumar 1 a la cuenta (puede ser en caso de cierta condición)
   }
```

Ejemplo:

Análisis
Requerimientos de datos:

Constantes del problema
RAD_SEGURA 0.466 /* nivel de radiacion seguro */

Dato del problema
double radiacion_inic /* nivel de radiacion inicial */

Salida del problema
int dia /* dias pasados desde el escape */
double nivel_radiacion /* nivel de radiacion en curso */

Variables adicionales
double radiacion_min; /* nivel de seguridad dividido por un factor de seguridad */

Diseño
Algoritmo inicial:
1. Inicializa dia a cero a radiacion_inic
2. Calcula el nivel de radiación seguro que termina el cálculo (rad. segura/factor seguridad)
3. Pregunta al usuario nivel de radiación inicial
4. Calcula e imprime el día u el nivel de radiación cada tres días en caso que la radiación exceda el nivel permitido.

Refinamiento del paso 4:
1.1. Inicializa nivel_radiacion a radiacion_ini
1.2. while nivel_radiacion > radiacion_min
1.2.1. Imprime el valor de dia, nivel_radiacion y el estado "Inseguro" o "Seguro"
1.2.2. Añadir 3 al valor de dia
1.2.3. Calcular nivel_radiacion para el siguiente periodo

Refinamiento del paso 4.2.1:
if nivel_radiacion > RAD_SEGURA
Imprime dia, nivel_radiacion y "Inseguro"
sino
Imprime dia, nivel_radiacion y "Seguro"

Figura 0.9: Explicación de cómo hacer el programa para calcular radiación

```
1 cuentagrandes=0;
2 for(cuenta = 1; cuenta <= veces; cuenta++) {
3 printf("Siguiente valor: ");
4 scanf("%f",&x);
5 if (x > corte) {
6 cuentagrandes++;
7 }
8 }
```

Cálculo de frecuencias de rechazos

```
1 /* ENUNCIADO
2 Leer por teclado datos de temperatura, presion y tiempo de
    tratamiento de una serie de muestras y sacar por pantalla el
    porcentaje que se rechaza por temperatura, presion o tiempo
```

```
/*
 * Calcula y muestra una tabla mostrando el nivel de radiacion
 * y seguridad.
 */
#include <stdio.h>

#define RAD_SEGURA     0.466  /* nivel de radiacion seguro */
#define FACT_SEGUR     10.0   /* factor de seguridad */

int main(void)
{
  int    dia;          /* dias pasados desde el escape */
  double radiacion_inic, /* nivel de radiacion despues del escape  */
         nivel_radiacion,  /* nivel de radiacion en curso */
         radiacion_min:  /* nivel de seguridad dividido por un factor de seguridad */

  /* Initializa dia y radiacion_min */
  dia = 0;
  radiacion_min = RAD_SEGURA / FACT_SEGUR;
  /* Pregunta al usuario el nivel de radiacion inicial */
  printf("Dar nivel de radiacion(en millirems)> ");
  scanf("%lf", &radiacion_inic);
  /* Muestra tabla */
  printf("\n  Dia  Radiacion  Estado\n       (milirems)\n");
  for (nivel_radiacion = radiacion_inic;
      nivel_radiacion > radiacion_min;
      nivel_radiacion /= 2.0) {
    if (nivel_radiacion > RAD_SEGURA)
      printf("  %3d    %9.4f   Inseguro\n", dia, nivel_radiacion);
    else
      printf("  %3d    %9.4f   Seguro\n", dia, nivel_radiacion);
    dia += 3;
  }
  return (0);
}
```

Figura 0.10: Programa solución para calcular el nivel de radiación

```
   fuera de rango. El rango de presion es de 60 a 70 kilos, el
   de temperatura es de 150 a 170 grados y el de tiempo es de 2
   a 2.5 segundos. La serie de muestras se concluira cuando el
   usuario teclee "fin"
ESQUEMA DE SOLUCIËN
-----Funcion principal
Ciclo contando fallos y total
        Si no se leen correctamente tres valores
                Terminar y escribir resultado
        Si esta fuera de rango de presion
                Contar otro fallo de presion
        Idem temperatura
                Idem tiempo
------Funcion de escritura de resultados
```

```
13 Calcular porcentaje referido a presion ---- usar funcion aparte
      para el calculo de porcentaje
14 Idem temperatura ----- vale la misma funcion, porque el calculo
      es el mismo
15 Idem tiempo ------ otra vez la misma, solo cambian los datos que
       se le pasan
16 Escribir los resultados
17 */
18 #include <stdio.h>
19 /* Límites máximo y mínimo de temperatura, presión y tiempo
20 Son datos del problema */
21 #define TEMAX 170
22 #define TEMIN 150
23 #define PRESMAX 70
24 #define PRESMIN 60
25 #define TIEMAX 2.5
26 #define TIEMIN 2
27 typedef float Grados, Kilos, Segundos;
28
29 void porcentaje(int parte, int total, float *resul)
30 /* Calcula qué porcentaje es parte sobre total */
31    {
32    *resul = (100. * parte) / total;
33    }
34
35 void escribe(int cuentem, int cuenpres, int cuentiem, int total)
36    {
37    float porcentem, porcenpres, porcentiem;
38    porcentaje(cuentem,total,&porcentem);
39    porcentaje(cuenpres,total,&porcenpres);
40    porcentaje(cuentiem,total,&porcentiem);
41    printf("Porcentaje de rechazos por temperatura: %g\n",
         porcentem);
42    printf("Porcentaje de rechazos por presión: %g\n", porcenpres
         );
43    printf("Porcentaje de rechazos por tiempo: %g\n", porcentiem)
         ;
44    }
45
46 void main()
47    {
48     /* Las cuentas de fallos por cada causa y el total*/
49     int cuentem, cuenpres, cuentiem, total;
50    Grados temp;
51    Kilos pres;
52    Segundos tiem;
53    for (cuentem = 0, cuenpres = 0, cuentiem = 0, total = 0; ;
```

```
54          /* Hay que poner los contadores a 0 al principio.
55          Las cuentas y condición de salida se controlan
               internamente
56          por lo que no se ponen en el for */)
57        {
58        printf("Dar temperatura, presión y tiempo: ");
59        if (scanf("%g %g %g",&temp, &pres, &tiem) < 3
60            /* El fallo en lectura es la condición de salida */)
61           {
62           break;
63           }
64        else
65           {
66           total++;
67           if (temp < TEMIN || temp > TEMAX)
68              {
69              cuentem++;
70              }
71           if (pres < PRESMIN || pres > PRESMAX)
72              {
73              cuenpres++;
74              }
75           if (tiem < TIEMIN || tiem > TIEMAX)
76              {
77              cuentiem++;
78              }
79           }
80        }
81     escribe(cuentem, cuenpres, cuentiem, total);
82     }
```

Este ciclo es para contar (pág. 106) y hay tres contadores y un acumulador para poner a 0 al principio. La condición de salida se controla dentro del ciclo, con una instrucción `break`y no hay nada que poner para el final de cada vuelta, porque los contadores se controlan dentro del ciclo; así pues los dos últimos campos del `for` quedan vacíos.

Acumulación de elementos leídos

Se trata de un recorrido leyendo (ver plantilla anterior) en el que se va acumulando (por ejemplo sumando, multiplicando . . .) los valores que se leen o algo obtenido de ellos. La variable en que se acumula tiene que ponerse a un valor inicial antes del ciclo. Por ejemplo, si se va a sumar será 0, si se va a multiplicar será 1, etc.

Plantilla:

```
acumulo=inicial;
   for(accion inicial; condicion de seguir; cambios) {
   scanf valor
```

```
   instrucciones de proceso
   acumular (puede ser en caso de cierta condición)
   }
```

Ejemplo:

```
1  sumat=0;
2  for (cuenta = 1; cuenta <= veces; cuenta++) {
3  printf("Siguiente valor: ");
4  scanf("%f",&x);
5  sumat += tan(x);
6  }
```

Cálculo de sueldos

```
1  /* ENUNCIADO
2  Leer cantidad de trabajadores y, de cada uno lo que gana por
      hora y cuántas horas ha trabajado y escribir sueldo neto
      individual de cada uno, la suma total de sueldos netos y el
      sueldo neto medio de la empresa.
3     Las horas extras (por encima de 40) se pagan vez y media más.
4     Se descuenta un 20% de retención.
5  ESQUEMA DE SOLUCIÓN
6  Leer cantidad de empleados
7  Poner total de sueldo a 0
8  Repetir contando empleados hasta llegar a la cantidad total
9           Leer paga y horas
10          Calcular sueldo bruto
11          Si pasa de 40 horas
12                  Añadirle un 50% más en las horas que pasen de 40
13          El sueldo neto es el 80% del bruto
14          Escribir el sueldo neto
15          Sumarlo al total
16 Escribir el total
17 Calcular la media
18 Escribir la media
19 */
20 #include <stdio.h>
21
22 /* Si la retención es el 20%, queda el 80% */
23 #define NETO 0.8
24 #define EXTRA 0.5 /* Es el exceso que se paga */
25 #define LIMHORAS 40
26
27 typedef float Euros;
28 /* Sólo se cuentan horas enteras */
29 typedef int Horas;
30
```

```
31 void main()
32    {
33    Euros total, medio;
34    Euros paga; /* Lo que gana por hora */
35    Euros sueldo; /* El sueldo de uno */
36    int empleados; /* Cuántos son */
37    Horas trabajo; /* Lo que ha trabajado uno */
38    int conta; /* Para llevar la cuenta de los que se hacen */
39
40   printf("Número de empleados: ");
41    scanf("%i", &empleados);
42    for (conta = 0, total = 0; conta < empleados; conta++
43         /* Antes de empezar se pone el total a 0 */)
44       {
45       /* El número de empleado le saca empezando en 1 */
46       printf("Dar paga horaria y horas de empleado %i: ",conta
            +1);
47       scanf("%g %i", &paga, &trabajo);
48       sueldo = paga * trabajo;
49       if (trabajo > LIMHORAS)
50          {
51          /* Hay que añadir un 50% más por las horas de exceso */
52          sueldo += EXTRA * paga * (trabajo - LIMHORAS);
53          }
54       sueldo *= NETO;
55       printf("Empleado %i gana %g\n", conta+1, sueldo);
56       total += sueldo;
57       }
58    printf("Total pagado: %g euros\n", total);
59    medio = total / empleados;
60    printf("Sueldo medio: %g\n", medio);
61    }
```

La plantilla es del tipo de recorrido acumulando (pág. 110) o contando (pág. 106). Contador y acumulador se inicializan a 0 y el primero se va aumentando hasta llegar al número de empleados.

Extremo

Es un caso de recorrido leyendo (ver plantillas anteriores). A la variable en que se va a guardar el extremo hay que darle un valor inicial opuesto a lo que se busca (por ejemplo, un valor muy grande si se busca un mínimo). También se puede dar como valor inicial el primero y empezar a comparar por el segundo.

Si no lo ves claro, en la figura 0.11 tienes una secuencia animada (si estás viendo el PDF con el visor de Adobe) que explica la idea.

Plantilla:

```
valor inicial de extremo
   for(accion inicial; condicion de seguir; cambios) {
```

Queremos sacar el máximo valor de una serie de ellos que nos van a presentar

Figura 0.11: Estrategia para sacar el máximo de una secuencia de valores

```
scanf valor
if es más extremo {
extremo=valor;
}
}
```

Ejemplo:

```
1 minimo=100000;
2 for (cuenta = 1; cuenta <= veces; cuenta++) {
3 printf("Siguiente valor: ");
4 scanf("%f",&x);
5 if (x<minimo) {
6 minimo=x;
7 }
8 }
```

Muestras de programas que requieren ciclos

Sumas de números que cumplan condiciones

Calcular todas las sumas de números obtenidas empezando por uno dado, terminando en otro dado y separados un salto entero que varíe entre dos límites dados también por el usuario. Sólo se sumarán los que sean múltiplos de cierto valor entero dado.

Ejemplo. Se empezarán en 2, se acabarán en 40 y los saltos irán desde 3 hasta 5. Sólo se sumarán los múltiplos de 7.

Caso de salto 3. Los números que salen son (subrayando los múltiplos de 7): 2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35, 38. Suma final: 14+35=49

Caso de salto 4: 2, 6, 10, 14, 18, 22, 26, 30, 34, 38. Suma final: 14

Caso de salto 5: 2, 7, 12, 17, 22, 27, 32, 37. Suma final: 7

Resolución. Como el problema nos parece inicialmente muy complejo, lo simplificamos, empezando con una versión inicial en que sólo se hace una suma con un sólo salto leído, y sin tener en cuenta si es múltiplo o no; se suman todos.

En este problema se van usando números que son consecutivos, aunque no de 1 en 1, sino según salto dado por teclado. En el programa resuelto de cuenta hacia adelante se presentan números consecutivos, así pues partimos de él.

El esquema era:

Leer inicio y final
Dar valor inicio al contador
repetir
  Escribir contador
  Aumentar contador
hasta que contador llegue a final

Vamos a suponer que el valor inicial y el cálculo van en una función. Vamos a modificarla.

En este caso no vamos de uno en uno, así que leemos el dato adicional y lo usamos para aumentar el contador, así que la modificación es:

Leer inicio y final
Dar valor inicio al contador
Repetir
    Escribir contador
    Aumentar contador
hasta que contador llegue a final

Dar valor inicio al contador
Repetir
    Escribir contador
    Aumentar contador *en salto*
hasta que contador llegue a final

Además no nos interesa ver los números, sino sumarlos, por lo que necesitaríamos otra variable para esa suma, luego lo cambiaríamos:

Dar valor inicio al contador
Repetir
    Escribir contador
    Aumentar contador en salto
hasta que contador llegue a final

Dar valor inicio al contador
Repetir
    Sumar contador al total acumulado
    Aumentar contador en salto
hasta que contador llegue a final

Por último tenemos en cuenta que el total debe empezar en 0, para que la primera suma salga correcta y que al final habrá que escribir el resultado, y con estos cambios llegamos a la solución final para nuestra versión inicial:

Poner a 0 el total acumulado
Dar valor inicio al contador
Repetir
    Sumar contador al total acumulado
    Aumentar contador en salto
hasta que contador llegue a final

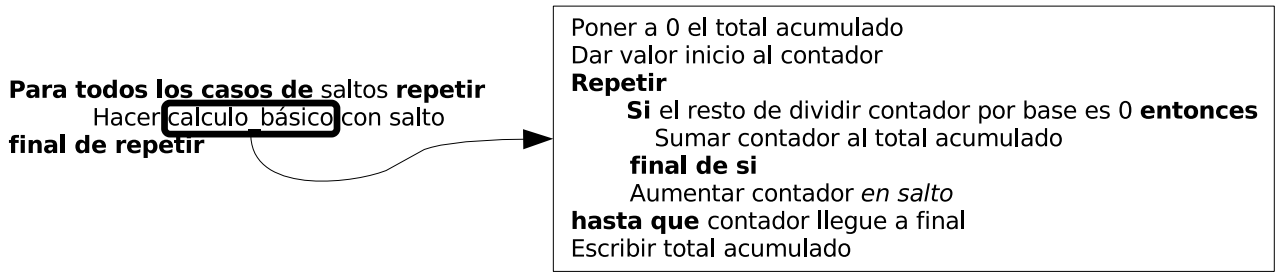
Dar valor inicio al contador
Repetir
    Sumar contador al total acumulado
    Aumentar contador *en salto*
hasta que contador llegue a final
Escribir total acumulado

A continuación ampliamos a una versión intermedia para tener en cuenta si son múltiplos o no, para lo cual leemos ese dato adicional (llamémoslo base) y, a la hora de sumar, lo usamos de forma que sólo sumamos si es múltiplo (o sea, su resto al dividir es 0). La función de cálculo habría que cambiarla:

Poner a 0 el total acumulado
Dar valor inicio al contador
Repetir
    Sumar contador al total acumulado
    Aumentar contador *en salto*
hasta que contador llegue a final
Escribir total acumulado

Poner a 0 el total acumulado
Dar valor inicio al contador
Repetir
    Si *el resto de dividir contador por base es 0* ***entonces***
        Sumar contador al total acumulado
    final de si
    Aumentar contador *en salto*
hasta que contador llegue a final
Escribir total acumulado

Por último nos planteamos llegar al enunciado completo en una versión final. Para empezar hay que leer los límites inferior y superior de los saltos y luego habrá que repetir toda la operación con distintos saltos. ¿Incluso el poner a 0 el total acumulado? Sí, porque empezamos la suma desde 0 con cada nuevo salto ¿Incluso escribir el total? Sí, porque queremos sacar el total cada vez. Por lo tanto, llamando a nuestra función anterior *calculobásico*, un primer borrador de la definitiva sería:



Por último especificamos la variación de salto. Debe empezar en su límite inferior hasta llegar al superior y subiendo de 1 en 1. Esta última modificación es:

Para todos los casos de saltos **repetir** ► **Para todos los** saltos *desde liminfsalto hasta limsupsalto* **repetir**
Hacer calculo_básico con salto
final de repetir

Hacer calculo_básico con salto
final de repetir

Ahora toca pasarlo a C Empezamos decidiendo las funciones. Como una función se corresponde con un bloque de proceso, vamos a meter en una el bloque de cálculo básico con un salto y el resto lo dejamos en `main`

Traduciendo a C las instrucciones de ese bloque nos queda:

```
1 acumulado=0;
2 for(contador=inicio;contador<=final;contador+=salto) {
3         if(contador % base == 0)  {
4                 acumulado+=contador;
5         }
6 }
7 printf("Total %i\n",acumulado);
```

Ahora tenemos que analizar cada variable para ver su tipo, si es argumento de entrada, de salida, o variable local.

Variable	Tipo	Argumento
acumulado	Sin decimales, puesto que los valores individuales no los tienen	Sólo se usa dentro de la función: variable local
contador	Sin decimales, porque recorre enteros	Sólo se usa dentro de la función: variable local
inicio	Sin decimales	Llega leída desde main, no se modifica en la función: argumento de entrada
final	Sin decimales	Llega leída desde main, no se modifica en la función: argumento de entrada
salto	Sin decimales	Llega desde main, no se modifica en la función: argumento de entrada
base	Sin decimales	Llega leída desde main, no se modifica en la función: argumento de entrada

Metiendo todo esto en el código llegamos a la función completa:

```
1 void bloquesalto(int inicio, int final, int salto, int base) {
2 int acumulado, contador;
```

```
3
4  acumulado=0;
5  for(contador=inicio;contador<=final;contador+=salto) {
6          if(contador % base == 0)  {
7                  acumulado+=contador;
8          }
9  }
10 printf("Total %i\n",acumulado);
11 }
```

En cuanto a `main`, le queda la lectura de datos y el ciclo que va variando salto y dentro del cual está la llamada a la función, para la que tendremos en cuenta los argumentos que hemos visto. Queda:

```
1  void main() {
2  int inicio, final, liminfsalto, limsupsalto, base, salto;
3
4  setlocale(LC_CTYPE,"");
5  printf("Inicio y final de los números: ");
6  scanf(" %i %i",&inicio,&final);
7  printf("Salto mínimo y máximo: ");
8  scanf(" %i %i",&liminfsalto,&limsupsalto);
9  printf("¿De quién tienen que ser múltiplos? ");
10 scanf(" %i",&base);
11 for(salto=liminfsalto; salto<=limsupsalto; salto++) {
12         bloquesalto(inicio,final,salto,base);
13 }
14 }
```

Si ponemos `includes` y lo juntamos tenemos el programa final:

```
1  #include <stdio.h>
2  #include <locale.h>
3
4  void bloquesalto(int inicio, int final, int salto, int base) {
5  int acumulado, contador;
6
7  acumulado=0;
8  for(contador=inicio;contador<=final;contador+=salto) {
9          if(contador % base == 0)  {
10                 acumulado+=contador;
11         }
12 }
13 printf("Total %i\n",acumulado);
14 }
15
16 void main() {
17 int inicio, final, liminfsalto, limsupsalto, base, salto;
18
```

```
19 setlocale(LC_CTYPE,"");
20 printf("Inicio y final de los números: ");
21 scanf(" %i %i",&inicio,&final);
22 printf("Salto mínimo y máximo: ");
23 scanf(" %i %i",&liminfsalto,&limsupsalto);
24 printf("¿De quién tienen que ser múltiplos? ");
25 scanf(" %i",&base);
26 for(salto=liminfsalto; salto<=limsupsalto; salto++) {
27         bloquesalto(inicio,final,salto,base);
28 }
29 }
```

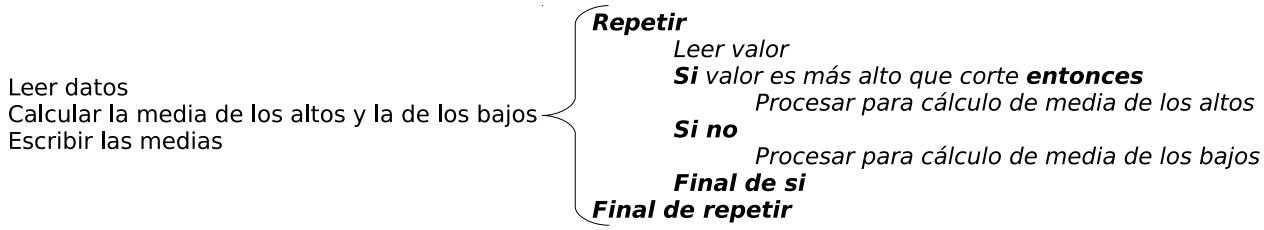
Medias

Dada una cantidad de valores, calcular por separado las medias de los que están encima y debajo de un número dado.

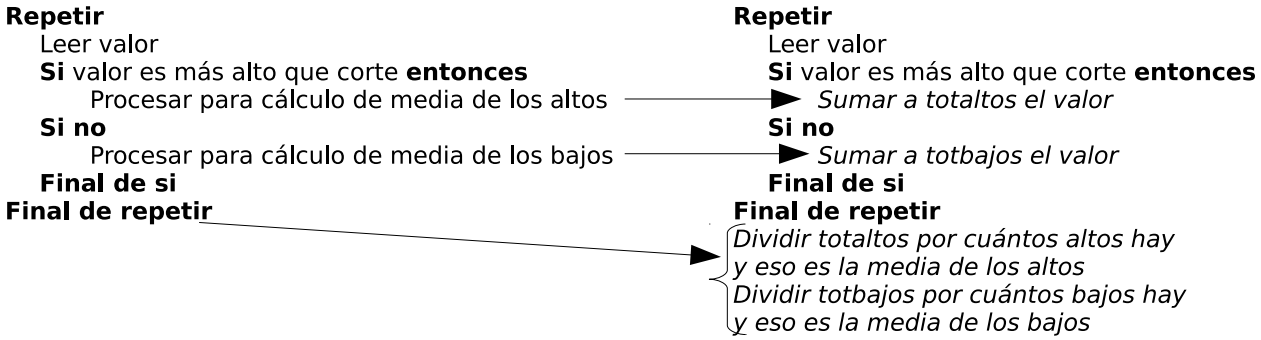
Resolución. Nuestra primera aproximación es la plantilla básica de leer datos, calcular y escribir resultados:

Leer datos
Calcular la media de los que están por encima y la de los que están por debajo
Escribir las medias

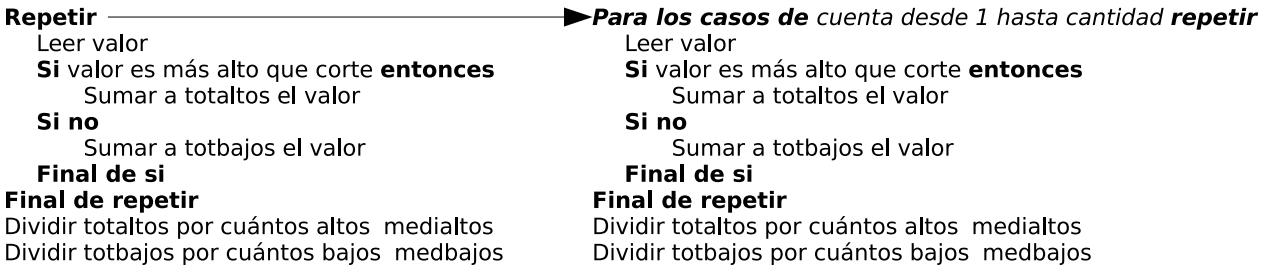
El punto gordo es cómo vamos a calcular las medias. En el programa esto puede ser una función aparte. Para refinar esto un poco más nos basamos en los elementos básicos de la programación que usamos: repetir un conjunto de instrucciones y meter en una variable un resultado. De momento planteamos que haremos el cálculo de las medias repitiendo alguna operación valor a valor. Será para la media de los altos si ese valor es superior al número de corte, y será para los bajos si es inferior.



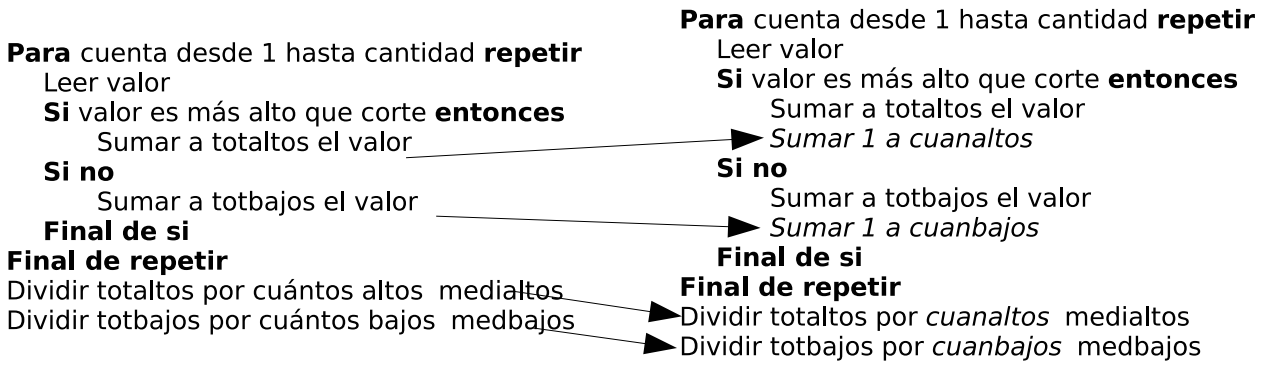
En nuestro siguiente refinamiento entramos a eso de "procesar". En este caso, recordamos que un elemento básico de la programación que hacemos es meter en una variable un resultado. La que queremos calcular es la media, para lo que usamos la definición de media: *la suma de todos los valores dividido por cuántos son*. La suma es una operación a ir repitiendo, porque se va sumando cada uno, mientras que la división se hace sólo una vez al final. Incluyendo esto tendríamos los cambios para sacar la función de cálculo:



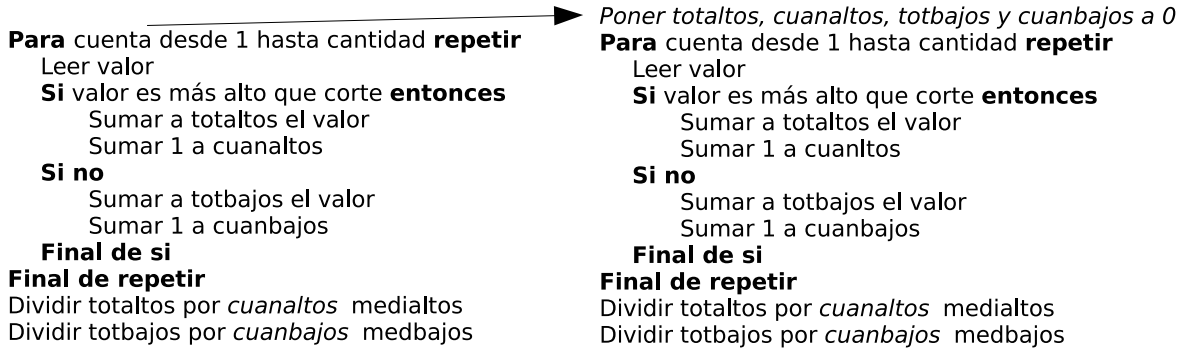
El programa tiene bastante estructura, pero quedan unos cuantos flecos. Hemos dejado abandonado lo de "leer datos". Si los valores se van leyendo en el ciclo de repetición, ¿qué otros datos hay que leer?: necesitamos algo para controlar el ciclo de repetición, porque si no se quedaría repitiendo para siempre. Una forma sencilla es pedir primero cuántos valores son y repetir contándolos desde 1 hasta la cantidad pedida. También estaría el valor de corte entre altos y bajos. Tendríamos los siguientes cambios en la función de calcular:



Por otra parte no hemos sacado cuántos altos y bajos hay. La forma más sencilla es cuando los tenemos identificados con el "Si valor es más alto . . . ". Quedaría hacer el siguiente cambio:



Por último, para que las sumas del ciclo funcionen bien hay que tener en cuenta cuál debe ser su valor inicial. Las medias son con la suma de los números y nada más, luego el valor inicial de esas sumas es 0. Las cuentas inicialmente deben estar también a 0. El algoritmo definitivo lo obtendremos con las modificaciones:



Si ahora lo traducimos a llegamos al siguiente conjunto de instrucciones para la función calcular:

```
totaltos=0;
cuanaltos=0;
totbajos=0;
cuanbajos=0;
for(cuenta=1; cuenta<=cantidad; cuenta++) {
  scanf(" %f",&valor);
  if (valor > corte) {
    totaltos += valor;
    cuanaltos++;
  } else {
    totbajos += valor;
    cuanbajos++;
  }
}
medialtos= totaltos/cuanaltos;
medibajos= totbajos/cuanbajos;
```

Pero esto es una función. Tenemos que analizar cada variable para ver su tipo, si es argumento de entrada, de salida, o variable local.

Variable	Tipo	Argumento
totaltos	Con decimales, puesto que los valores individuales pueden tenerlos	Sólo se usa dentro de la función: variable local
cuanaltos	Sin decimales, es un recuento	Sólo se usa dentro de la función: variable local
totbajos	Con decimales	variable local
cuanbajos	Sin decimales	variable local
cuenta	Sin decimales, es un recuento	Sólo se usa dentro de la función: variable local
cantidad	Sin decimales, es una cantidad	Llega leída desde main, no se modifica en la función: argumento de entrada
valor	Con decimales	Sólo se usa dentro de la función: variable local
corte	Con decimales	Llega leída desde main, no se modifica en la función: argumento de entrada
medialtos	Con decimales	Se le da valor en la función y eso se escribe después en main: argumento de salida
medibajos	Con decimales	argumento de salida

Metiendo todo esto en el código llegamos a la función completa:

```
1  void calmedias(int cantidad, float corte, float *medialtos,
       float *medibajos) {
2  float totaltos, totbajos, valor;
3  int cuanaltos, cuanbajos, cuenta;
4
5  totaltos=0;
6  cuanaltos=0;
7  totbajos=0;
8  cuanbajos=0;
9  for (cuenta=1; cuenta<=cantidad; cuenta++) {
10 scanf(" %f",&valor);
11 if (valor > corte) {
12 totaltos += valor;
13 cuanaltos++;
14 }       else {
15 totbajos += valor;
16 cuanbajos++;
17 }
18 }
19 *medialtos= totaltos/cuanaltos;
20 *medibajos= totbajos/cuanbajos;
21 }
```

Nos falta la parte de `main` que tiene sólo la lectura de datos, llamada a la función

calmedias y escritura de resultados. Para la llamada tendremos en cuenta los argumentos que hemos visto. Queda:

```
1  void main()
2  {
3  float corte; /* Límite de separación entre grupos */
4  float medialtos, medibajos;
5  int cantidad;
6
7  setlocale(LC_CTYPE,"");
8  printf("Dar cantidad de valores: ");
9  scanf(" %i", &cantidad);
10 printf("Dar valor de separación: ");
11 scanf(" %f", &corte);
12 calmedias(cantidad,corte,&medialtos,&medibajos);
13 printf("Media de altos: %f\nMedia de bajos: %f\n",
14         medialtos, medibajos);
15 }
```

Si lo ponemos todo junto, llegamos al programa siguiente

Cálculo de medias separadas de valores altos y bajos

```
1  /* ENUNCIADO
2  Escribir un programa para obtener el peso medio de los gordos y
      el de los delgados en un grupo de personas. Se consideran
      gordos los que queden por encima de un peso que se pedira al
      inicio. Se consideran delgados los que queden por debajo. Los
       pesos podran estar desordenados.
3  ESQUEMA DE SOLUCION
4  Leer el cantidad y el valor de separacion entre gordos y flacos
5  ----------Funcion para calcular las medias
6  Poner cuentas de gordos y flacos a 0
7  Poner totales de pesos de gordos y flacos a 0
8  Leer peso, hasta que se cumpla la cuenta
9          Si es mayor que la separacion
10                 Contarle como gordo
11                 Sumar su peso al de los gordos
12         Si no,
13                 Contarle como flaco
14                 Sumar su peso al de los flacos
15 Calcular las dos medias
16 ------------------------------------------
17 Escribirlas
18 */
19 #include <stdio.h>
20 #include <locale.h>
21
```

```
22 void calmedias(int cantidad, float corte, float *medialtos,
      float *medibajos) {
23 float totaltos, totbajos, valor;
24 int cuanaltos, cuanbajos, cuenta;
25
26 totaltos=0;
27 cuanaltos=0;
28 totbajos=0;
29 cuanbajos=0;
30 for (cuenta=1; cuenta<=cantidad; cuenta++) {
31 scanf(" %f",&valor);
32 if (valor > corte) {
33 totaltos += valor;
34 cuanaltos++;
35 }       else {
36 totbajos += valor;
37 cuanbajos++;
38 }
39 }
40 *medialtos= totaltos/cuanaltos;
41 *medibajos= totbajos/cuanbajos;
42 }
43 void main()
44 {
45 float corte; /* LÝmite de separaci¾n entre grupos */
46 float medialtos, medibajos;
47 int cantidad;
48
49 setlocale(LC_CTYPE,"");
50 printf("Dar cantidad de valores: ");
51 scanf(" %i", &cantidad);
52 printf("Dar valor de separaci¾n: ");
53 scanf(" %f", &corte);
54 calmedias(cantidad,corte,&medialtos,&medibajos);
55 printf("Media de altos: %f\nMedia de bajos: %f\n",
56         medialtos, medibajos);
57 }
```

Se usa un plantilla similar a las de contar (pág. 106) y acumular (pág. 110).

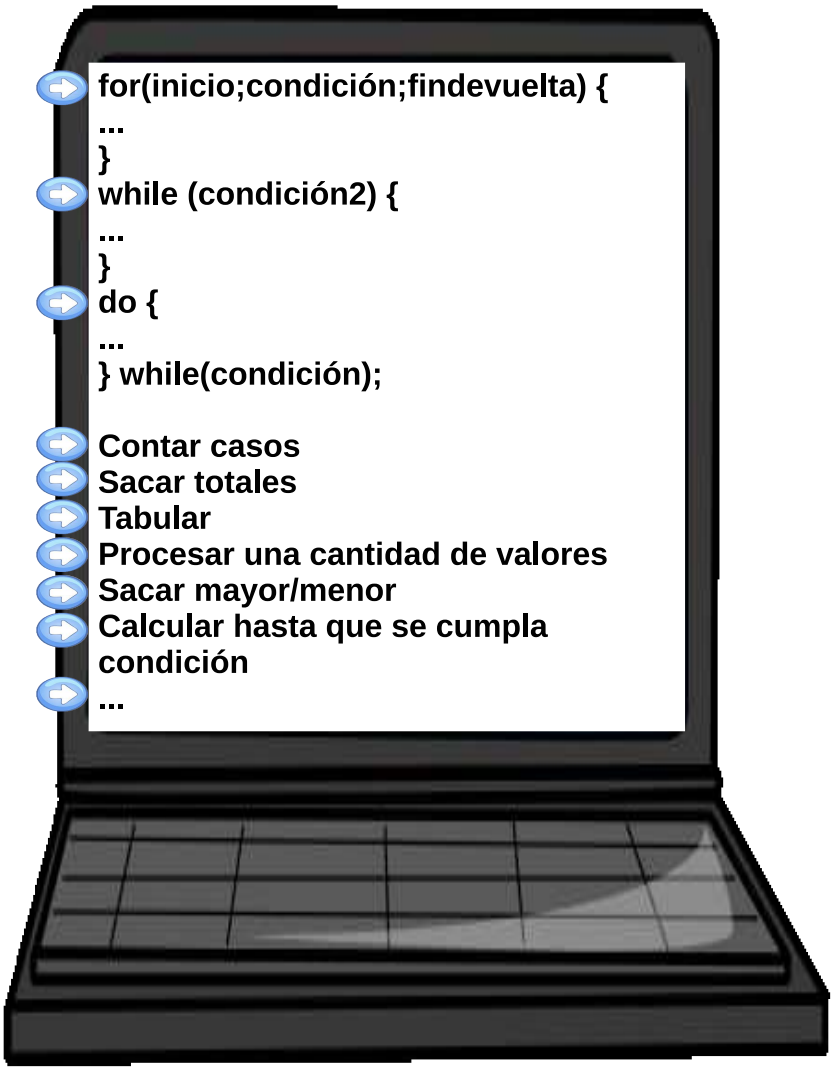
Cálculo de puntuación máxima en diana

```
1 /* ENUNCIADO
2 En un concurso de dardos, un jugador tiene una cantidad de
     intentos y se le dará al final la puntuación
3 correspondiente al mejor de sus intentos. Hacer un programa que
     lea de cuántos intentos dispone y luego la puntuación
     obtenida en cada uno y escriba al final la puntuación que se
```

```
   le adjudica. Si en algún intento logra el 10, como eso no se
   puede superar, terminará ahí.
ESQUEMA DE SOLUCIÓN
Leer cantidad de intentos
-----Función para leer puntos de cada intento y asignar
   puntuación definitiva
Poner puntuación máxima a 0
Repetir contando intentos hasta la cantidad indicada
        Pedir y leer la puntuación de ese intento
        Si es mayor que la máxima
                La máxima pasa a ser la de este intento
        Si ha sacado 10
                Terminar
-------------------------------------------------------------------------

Escribir la puntuación máxima alcanzada
*/
#include <stdio.h>
#define MAXIMO 10

void diana(int intentos, int *maximo)
   {
   int conta; /* Para llevar la cuenta de los que se hacen */
   int puntos; /* De un intento */
   for (conta = 0, *maximo = 0; conta < intentos; conta++)
      {
      /* El nmero de intento le saca empezando en 1 */
      printf("Dar puntuacion de intento %i: ",conta+1);
      scanf("%i", &puntos);
      if (puntos > *maximo)
         {
         *maximo = puntos;
         }
      if (puntos == MAXIMO)
         {
         return
                /* Tambin podra ser un break */;
         }
      }
   }

void main()
   {
   int intentos, maxpun;
   printf("Nmero de intentos: ");
   scanf("%i", &intentos);
   diana(intentos,&maxpun);
```

```
47    printf("Puntuacin mxima: %i\n", maxpun);
48    }
```

Se aplica la plantilla de recorrido buscando extremo (pág. 112), con el añadido de que si se alcanza el máximo posible ya no merece la pena seguir y se termina.



Ejemplos resueltos: Ciclos

Escribir una cuenta hacia delante

```
#include <stdio.h>
void cuenta(int lim)
   {
   int voy;
   for (voy=1; voy<=lim; voy++)
      {
      printf("%i ",voy);
      }
   }
void main()
   {
   int lim;

   printf("Limite de cuenta: ");
   scanf("%i",&lim);
   cuenta(lim);
   }
```

El contador se pone primero al valor de salida, la condición es que no pase del límite y en cada vuelta sube de uno en uno. La instrucción a repetir es la de escritura.

Escribir una cuenta hacia atrás

```
#include <stdio.h>
void cuenta(int ini)
   {
   int conta;
   for (conta = ini; conta >= 0; conta--)
      {
      printf("%i ",conta);
      }
   }
void main()
   {
   int ini;

    printf("Principio de cuenta atras: ");
   scanf("%i",&ini);
   cuenta(ini);
   }
```

Es análogo al anterior, pero esta vez la condición de repetición es que el valor sea positivo.

Tabla con conversión de grados a radianes

```
/* ENUNCIADO
Teniendo en cuenta que pi radianes son 180 grados, escribir una
   tabla de equivalencia entre grados y radianes. La tabla
```

```
   tendrá dos columnas, la primera para los grados y la segunda
   para los radianes. Empezará en 0 grados y acabará en 360 El
   incremento de grados entre filas se habrá leído previamente
   de teclado.
ESQUEMA DE SOLUCIÓN
Leer salto de grados entre filas
--------Función que escribe la tabla
Ciclo contando grados, empezando en 0 y acabando en 360,
   subiendo cada vez el salto pedido
         Escribir los grados y los radianes
*/
#include <stdio.h>
#define FACTOR (3.1416/180)
#define TOPE 360
typedef float Grados;

void tabla(Grados intervalo)
   {
   Grados angulo;
   printf("Grados\tRadianes\n");
   for (angulo = 0; angulo < TOPE; angulo += intervalo)
      {
      printf("%6g\t%g\n",angulo,angulo*FACTOR);
      }
   }

void main()
   {
   Grados intervalo;
   printf("Dar intervalo: ");

   if (scanf("%g",&intervalo)  == 1)
      {
      tabla(intervalo);
      }
   else
      {
      printf("Error de lectura\n");
      }
   }
```

El contador esta vez es el ángulo y, en vez de subir uno a uno, va a un salto cada vez. Lo que se repite es escribir el ángulo y su equivalencia. Se usa una plantilla del tipo de recorrer calculando (pág. 106)

Cálculo de calado en un canal

```
/* ENUNCIADO
```

```
2  Hacer un programa que ayude al usuario a calcular la profundidad
       de agua en un canal de seccion rectangular. Los datos son:
   la anchura del canal, su inclinacion, su rugosidad y el
   caudal de agua. Se usara la formula de Manning:
3       1   1/2 2/3
4  v = - I   R
5       n
6  En palabras, la velocidad (v) es la raiz cuadrada de la
   inclinacion (I), por el radio hidraulico (R) elevado a dos
   tercios y dividido por el coeficiente de rugosidad (n)
7  El radio hidraulico es el area de seccion de agua divido por el
   perimetro de canal con el que esta en contacto
8  El caudal es la velocidad multiplicada por la seccion de agua
9  Por mucho que se manipule esa formula, no se puede despejar la
   profundidad de agua, que sale en la seccion y en el perimetro
       mojado. Para averiguar la solucion el programa lo que hara
   es pedir repetidamente tanteos al usuario y calcular el
   caudal que sale con ese tanteo. Cuando la diferencia entre
   ese caudal y el que tiene que salir de verdad este por debajo
       del 3%, se dara por bueno.
10
11 ESQUEMA DE SOLUCION
12 Pedir y leer los datos
13 --------------------Funcion de comprobacion de tanteos
14 Repetir hasta que el porcentaje de error sea menor que 3
15         Pedir calado (profundidad)
16         Calcular el caudal usando la formula: la seccion es el
              ancho por la profundidad y el perimetro mojado es el
              ancho mas dos veces la profundidad
17         Calcular el porcentaje de error de caudal
18 ------------------------------------------------------
19 Escribir el calado/profundidad aceptada
20 */
21 #include <stdio.h>
22 #include <math.h> /* Para la función pow */
23 #define MAXERROR 3
24 typedef float Metros, Pendiente, Rugosidad, Metros3, Metros2;
25
26 void calcula(Metros ancho, Pendiente inclin,
27  Rugosidad n, Metros3 caudal, Metros *calado)
28    {
29    Metros2 area;
30    Metros radio;
31    Metros3 caudaprox; /* Para los tanteos */
32    float porcerror; /* Porcentaje de error en el tanteo */
33    do
34       {
```

```
35        /* La fórmula da caudal en función de calado,
36        así que el usuario va dando calados hasta que
37        sale muy cerca del caudal real */
38        printf("Dar calado: ");
39        scanf("%g", &*calado);
40        area = ancho * (*calado);
41        radio = area / (ancho + 2*(*calado)); /* Definición de
              radio hidráulico */
42        caudaprox = area * pow(radio,0.666)*sqrt(inclin)/n;
43        porcerror = (fabs(caudaprox - caudal) / caudal) * 100;
44        printf("Caudal que sale: %g\n", caudaprox);
45        }
46     while (porcerror > MAXERROR);
47     }
48
49  void main()
50     {
51     Metros ancho, calado;
52     Pendiente inclin;
53     Rugosidad n;
54     Metros3 caudal;
55     printf("Dar ancho: ");
56     scanf("%g",&ancho);
57     printf("Dar inclinación y coeficiente de rugosidad: ");
58     scanf("%g %g", &inclin, &n);
59     printf("Dar caudal: ");
60     scanf("%g", &caudal);
61     calcula(ancho, inclin, n, caudal,&calado);
62     printf("Calado final: %g\n",calado);
63     }
```

Las fórmulas se siguen aplicando mientras el error no sea suficientemente bajo.

Sacar números primos

```
1  /*ENUNCIADO
2  Hacer un programa que saque tantos numeros primos como se le
       pidan, a partir del 2 en adelante.
3  ESQUEMA DE SOLUCIËN
4  Pedir la cantidad de primos
5  ------------------------Funcion de irlos sacando
6  Recorrer los numeros desde el 2 en adelante, hasta que la cuenta
       de primos llegue a la cantidad pedida
7          Recorre los numeros previos al que se esta comprobando,
               desde 2 hasta su raiz cuadrada
8                  Si uno de los previos es divisor exacto del
                       comprobado, salir del ciclo
9          Si se ha llegado probando hasta la raiz cuadrada
```

```
10                  Escribirlo
11                  Incrementar la cuenta de primos
12  */
13  #include <stdio.h>
14  #include <math.h> /* Para sqrt que hace la raíz cuadrada */
15  #define EMPIEZA 2
16
17  void sacaprimos(int cantidad)
18     {
19     /* Para llevar la cuenta de los primos que se sacan */
20     int contap;
21     int numero; /* El número por el que se va */
22     int divisor, limmax; /* Para ir probando divisores */
23     for (numero = EMPIEZA, contap = 0; contap < cantidad; numero
          ++)
24        {
25        for (divisor = EMPIEZA, limmax = sqrt((float)numero);
26              divisor <= limmax; divisor++
27              /* Para ver si es primo  va probando si
28              tiene divisores
29              hasta su raíz cuadrada */)
30           {
31           if (numero % divisor == 0
32                /* sale prematuramente
33                cuando encuentra un divisor */)
34              {
35              break;
36              }
37           }
38        if (divisor > limmax
39             /* entonces es que no encontró divisores */)
40           {
41           printf(" %i",numero);
42           contap++;
43           }
44        }
45     }
46
47  void main()
48     {
49     int cantidad;
50     printf("Cuntos primos?");
51     scanf("%i", &cantidad);
52     sacaprimos(cantidad);
53     }
```

Hay dos ciclos, uno dentro de otro. El exterior, para contar cuántos se van sacando tiene un

estilo habitual de cuenta (pág. 104). El interior es para probar divisores, de uno en uno, hasta llegar a la raíz cuadrada del número, ya que, si no ha habido divisores hasta ahí, ya no los va a haber; el cálculo inicial del límite se ha incluido en el primer campo del `for` En este caso hay una condición adicional de salida que se comprueba dentro del ciclo: que sea divisor; si es el caso, se termina con la instrucción `break`

En la figura 0.12 tienes una secuencia animada (si estás viendo el PDF con el visor de Adobe) del programa para el caso de que se le pidan 3 primos. Te va marcando la instrucción que va haciendo y a la derecha te saca cómo quedan las variables y lo que va saliendo en pantalla (lo que teclea el usuario queda marcado con texto rojo). Con los botones de control de la derecha puedes aumentar o reducir la velocidad y con los de la izquierda puedes parar, reanudar, ir paso a paso o volver al principio. La pones en marcha con el botón ▷

Número del Tarot

En las páginas siguientes tienes explicado este programa cuya principal dificultad consiste en sumar las cifras de un número.

```
Recorrer los numeros desde el 2 en adelante, hasta que la cuenta de primos llegue a
        Recorre los numeros previos al que se esta comprobando, desde 2 hasta su ra
                Si uno de los previos es divisor exacto del comprobado, salir del c
        Si se ha llegado probando hasta la raiz cuadrada
                Escribirlo
                Incrementar la cuenta de primos
*/
#include <stdio.h>
#include <math.h> /* Para sqrt que hace la raíz cuadrada */
#define EMPIEZA 2

void sacaprimos(int cantidad)
   {
   /* Para llevar la cuenta de los primos que se sacan */
   int contap;
   int numero; /* El número por el que se va */
   int divisor, limmax; /* Para ir probando divisores */
   for (numero = EMPIEZA, contap = 0; contap < cantidad; numero++)
      {
      for (divisor = EMPIEZA, limmax = sqrt((float)numero);
           divisor <= limmax; divisor++
           /* Para ver si es primo  va probando si
           tiene divisores
           hasta su raíz cuadrada */)
         {
         if (numero % divisor == 0
             /* sale prematuramente
             cuando encuentra un divisor */)
            {
            break;
            }
         }
      if (divisor > limmax
          /* entonces es que no encontró divisores */)
         {
         printf(" %i",numero);
         contap++;
         }
      }
   }

void main()
   {
   int cantidad;
   printf("Cuntos primos?");
   scanf("%i", &cantidad);
   sacaprimos(cantidad);
   }
```

Pantalla

Variable	Valor

Figura 0.12: Animación del funcionamiento del programa de sacar números primos

```
/* Averigua el numero del Tarot: se suma dia, mes y año de nacimiento
y luego se suman las cifras del número obtenido, repetidamente hasta
obtener un valor inferior a 10
  * Tomado del libro “C-C++: curso de programación” de F. J. Ceballos */

#include <stdio.h>

void main()
{
 int dd, mm, aaaa, ddo, resto, tarot;

 printf("Si tecleas tu fecha de nacimiento, te calculo tu número de Tarot\n");
 do
 {
  printf("Dia:  ");
  scanf("%d", &dd);
 }
 while (dd < 1 || dd > 31); // leer el dia mientras no sea correcto
                            // (se supone que los dias 29, 30 y 31 son posibles)
 do
 {
  printf("Mes:  ");
  scanf("%d", &mm);
 }
 while (mm < 1 || mm > 12); // leer el mes mientras no sea correcto

 do
 {
  printf("Anho:  ");
  scanf("%d", &aaaa);
 }
 while (aaaa < 0 || aaaa > 9000); // leer el año
```

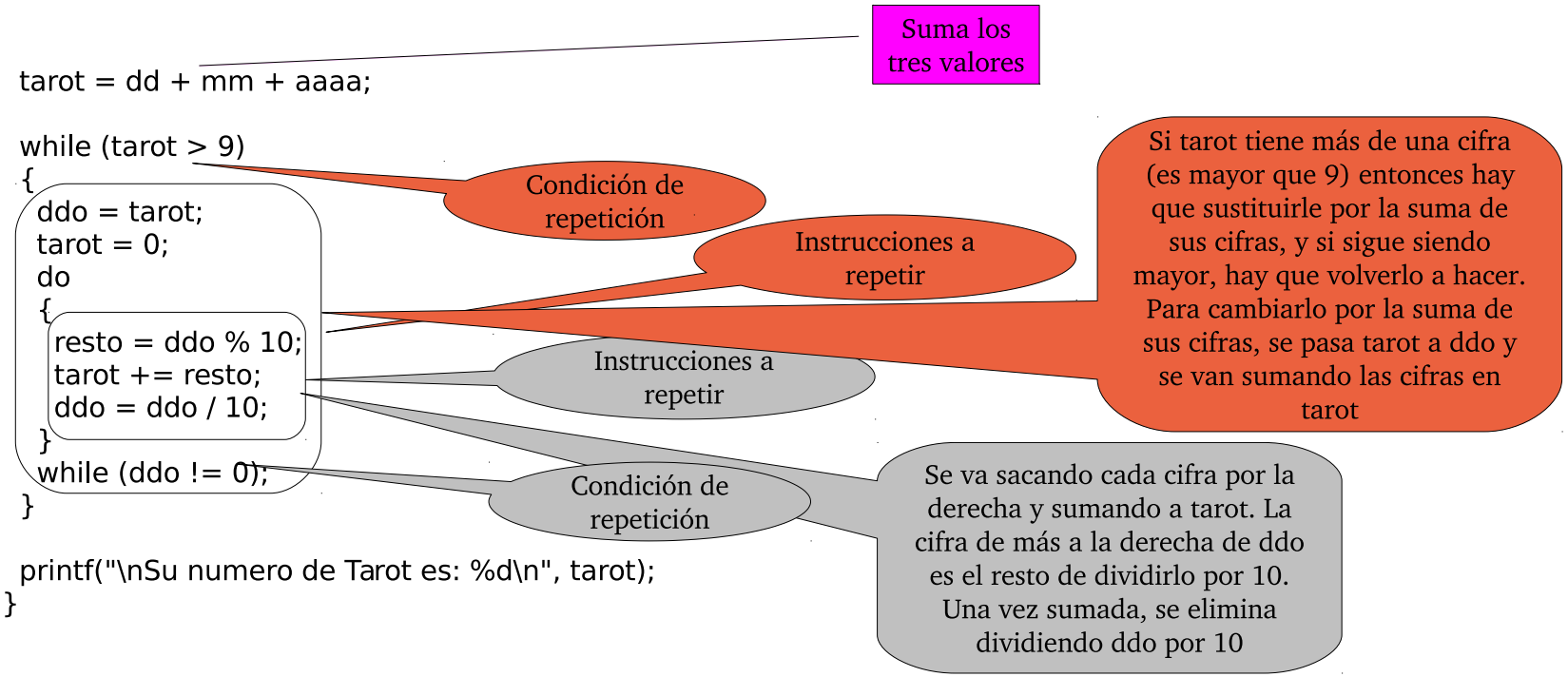
Instrucciones a repetir

Condición de repetición

Si el valor leído (dd) es menor que 1 o mayor que 31, se vuelve a repetir el ciclo, es decir, se vuelve a pedir y a leer uno nuevo. Es un ciclo do-while, porque no se puede comprobar antes de haberlo leído, hay que comprobarlo después, al final del bloque.

Lo mismo para los meses

Lo mismo para los años; sólo admite entre 0 y 9000


tarot = dd + mm + aaaa;
Suma los tres valores
while (tarot > 9)
{
ddo = tarot;
tarot = 0;
do
{
resto = ddo % 10;
tarot += resto;
ddo = ddo / 10;
}
while (ddo != 0);
}
printf("\nSu numero de Tarot es: %d\n", tarot);
}
Condición de repetición
Instrucciones a repetir
Si tarot tiene más de una cifra (es mayor que 9) entonces hay que sustituirle por la suma de sus cifras, y si sigue siendo mayor, hay que volverlo a hacer. Para cambiarlo por la suma de sus cifras, se pasa tarot a ddo y se van sumando las cifras en tarot
Instrucciones a repetir
Condición de repetición
Se va sacando cada cifra por la derecha y sumando a tarot. La cifra de más a la derecha de ddo es el resto de dividirlo por 10. Una vez sumada, se elimina dividiendo ddo por 10

Datos estructurados

Llamamos datos estructurados a aquellas variables que sirven para manejar más de un valor simple, por ejemplo, varios números o letras. Como en C sólo se opera con un valor simple cada vez, hay que poder elegir el valor que queremos dentro del conjunto. Si el conjunto de valores está ordenado con un sólo valor de posición, le vamos a llamar lista. Las listas de caracteres (textos) tienen algunos tratamientos especiales en C, así que las estudiaremos separadamente. Si hay dos valores de posición (fila y columna) le llamamos tabla.

C: Listas

¿En qué consisten las listas, vectores o arrays?

¿Cómo se declaran?

¿Cómo se operan?

¿Cómo se pasan a funciones?

Recursos en el aula virtual de listas

Teoría

Aparte de estos apuntes

Ejercicios iniciales de instrucciones concretas

Para comprobar que has interiorizado la teoría
Y recuerda las cuestiones de autoevaluación que tienes aquí en el apartado III

Ejemplos explicados

Con la solución y explicaciones
Y también en estos apuntes, en el apartado III

Ejemplos activos

Visualizaciones de la ejecución paso a paso de un programa

Ejemplos con solución

La solución sin o con pocas explicaciones
Y también en estos apuntes, en el apartado III

Ejercicios con pistas

Sin memorización de por dónde vas

Ejercicios propuestos

Sin la solución

Controles y exámenes de otros años

Algunos con solución, como ejemplos resueltos, y otros sin ella, como ejercicios propuestos

Las listas son conjuntos de valores del mismo tipo que se manejan homogéneamente. Vienen dadas porque en el problema hay una serie de valores. Son ineludibles cuando la cantidad de elementos no se conoce a la hora de hacer el programa. Una lista, vector o *array* se puede definir como un conjunto de datos del mismo tipo a los que se da un nombre común y a los que se accede a través de un índice. Se declaran según la sintaxis:

```
TipoDato variable_array[numero_elementos];
```

Algunos ejemplos son:

```
1 int numeros[20];
2 float temperaturas[100];
```

En un vector de N elementos, el primero se referencia con el índice 0 y el último con el índice N-1. La inicialización se puede realizar de dos modos, si es que interesa:

```
1 int numeros[2]={0,1};
2 int numeros[]={0,1};
```

En el segundo caso, el compilador asume un array de 2 elementos.

Se debe recordar que el procesamiento de arrays se debe realizar elemento a elemento, utilizando el operador índice de array, `[indice]`**, que es usado para acceder a los distintos elementos que componen un array.**

Listas como parámetros de una función

Una lista o array (o vector) se pasa a una función especificando su nombre sin corchetes. En C, los arrays siempre pueden ser de entrada y de salida (**no hace falta el operador &**).

En la cabecera el argumento correspondiente al array se escribe con un par de corchetes cuadrados con su tamaño, como una declaración. En realidad es optativo el tamaño pero, como veremos, en las tablas no todas las dimensiones son optativas, lo que puede crear confusión; por ello se recomienda dar siempre el tamaño.

Llamada

Hay varias posibilidades. Una plantilla que valga para todos los casos puede ser:

```
nombre(varentra, &varsale,lista, tabla,...);
```

Ejemplo:

```
1 leervector(tamanho, vector);
```

Cabecera

La llamada debe ser conforme a la cabecera. Para la llamada anterior la plantilla sería:

```
void nombre(tipo varentra, tipo *varsale,tipo lista[tamaño], tipo
   tabla[filas][columnas],...) {
   ...
   }
```

Ejemplo:

```
1 void leervector(int tamanho, float vector[tamanho]) {
2 ...
3 }
```

Funciones disponibles para listas

Función memcmp (<string.h>)

Compara los primeros n valores de la lista s1 con los primeros n de s2 (ambas son del mismo tipo). La función retorna cero si la lista s1 es igual que la s2.

Plantilla:

if (**memcmp(lista1,lista2, cantidaddeelementos*sizeof(tipo))** ==0)

Ejemplo:

if (**memcmp(s1,s2, n*sizeof(int))** ==0)

Función memcpy (<string.h>)

Copia los primeros n valores de la lista s2 a s1, ambas de cierto tipo.

Plantilla:

```
memcpy(destino, origen, cantidaddeelementos*sizeof(tipo));
```

Ejemplo:

```
memcpy(s1, s2, n*sizeof(float));
```

Función memset (<string.h>)

Está pensada para caracteres, pero puede usarse para poner a 0 cada uno de los primeros n valores de la lista s, indicando su tipo.

Plantilla:

```
memset(lista, 0, cantidaddeelementos*sizeof(tipo));
```

Ejemplo:

```
memset(s, 0, n*sizeof(int));
```

Función qsort (<stdlib.h>)

Esta función ordena una lista de n objetos. Uno de los argumentos es el tamaño de cada elemento de la lista, que se obtiene de su tipo con la función `sizeof`. El contenido de la lista se ordena de forma ascendente usando la función de comparación apuntada por `comparar`, que debemos incluir aparte en el programa, la cual es llamada con dos argumentos de salida: los objetos a ser comparados. La función debe retornar un entero menor, igual, o mayor que cero si el primer objeto se considera, respectivamente, menor, igual, o mayor que el segundo.

Plantilla:

```
qsort(lista, cantidaddeelementos, sizeof(tipo), &funcióndecomparar);
```

Ejemplo:

```
qsort(lista, n, sizeof(int), &comparar);
```

La ordenación se detalla en la plantilla correspondiente (pág. 141)

Plantillas: Listas

Declaración

Plantilla:

```
tipovalores nombrelista[tamaño];
```

Ejemplo:

```
1 float numeros[CANTIDAD];
```

Uso

Elemento individual

Plantilla:

```
...nombrelista[posicion]...
```

Ejemplo:

```
1 scanf("%f",&numeros[0]);
```

Lista completa

Plantilla:

```
..nombrelista..
```

Ejemplo:

```
1 memcpy(numer2,numeros1,sizeof(numeros1));
```

Lectura

Para el caso de lecturas típicas se ha preparado en un fichero extra una instrucción (técnicamente es una *macro*), que lee consecutivamente la serie de valores.

Plantilla:

```
lelisnu(formato,lista,cantidaddevalores);
```

Ejemplo:

```
1  lelisnu("%f",alturas,cuantas);
```

Escritura

Para el caso de escrituras típicas se ha preparado una instrucción (técnicamente es una *macro*), que escribe consecutivamente la serie de valores. Conviene tener en cuenta en el formato la separación entre valores.

Plantilla:

```
eslisnu(formato,lista,cantidaddevalores);
```

Ejemplo:

```
1  eslisnu("%f ",alturas,cuantas);
```

Recorridos por lista

El proceso a hacer con cada elemento de la lista puede ser cualquiera. En el ejemplo se presenta una escritura de lista. Si se usara `scanf` en lugar de `printf`, se tendría una lectura de lista.

Plantilla:

```
for(posicion = 0; posicion < tamaño; posicion++) {
   instrucciones trabajando lista[posicion]
   }
```

Ejemplo:

```
1 for(posicion = 0; posicion < cuantos; posicion++){
2 printf("%i",valores[posicion]);
3 }
```

Cuenta de elementos de lista que cumplan condición

Es análogo al de cuenta genérico (pág. 106), pero en vez de leer el valor, es un elemento de una lista. También se puede considerar como un recorrido por lista (ver plantilla anterior), al que se añade el efecto de contar.

Plantilla:

```
cuenta=0;
   for(posicion = 0; posicion < tamaño; posicion++) {
   if (lista[posicion] cumple condición) {
   cuenta++;
   }
   }
```

Ejemplo:

```
1 cuentapeq=0;
2 for(posicion = 0; posicion < cuantos; posicion++) {
3 if (valores[posicion] < pequenho) {
4 cuentapeq++;
5 }
6 }
```

Acumulación de elementos de lista

Es análogo al genérico de acumulación (pág. 110), pero en vez de leer el valor, es un elemento de una lista. También se puede considerar como un recorrido por lista (ver plantilla anterior), al que se añade el efecto de acumular.

Plantilla:

```
valor inicial de acumulación
   for(posicion = 0; posicion < tamaño; posicion++) {
   acumular respecto a lista[posicion]
   }
```

Ejemplo:

```
1 producto=1;
2 for(posicion = 0; posicion < cuantos; posicion++) {
3 producto *= valores[posicion];
4 }
```

Ordenación

Los problemas de ordenación aparecen con cierta frecuencia. Se van a presentar tres formas de abordarlo:

1. Usando unas macros ya preparadas para casos simples
2. Con la función para ello que trae el lenguaje C
3. Resolviendo la ordenación directamente como problema de programación

Casos elementales: macro

Se ha preparado unas instrucción (técnicamente una *macro*) para el caso más simple de ordenar listas, de menor a mayor.

Plantilla:

```
ordena(lista,numerodelementos);
```

Ejemplo:

```
1 ordena(veces,cantidad);
```

Función de ordenación incluida en C

El lenguaje C trae implementada una función para ordenar (`qsort`) Requiere que los elementos estén contiguos. Pondremos la instrucción en el punto donde queramos la ordenación y una función adicional de comparación definida previamente.

El esquema de la función y de la instrucción que iría donde se quiera ordenar sería:

```
int nombrefun(const void *e1, const void *e2) {
   const tipo *el1, *el2;
   el1=(tipo *)e1;
   el2=(tipo *)e2;
   if ((*el1) tiene que ir antes que (*el2)) return enteronegativo;
   if ((*el2) tiene que ir antes que (*el1)) return enteropositivo;
   return 0;
   }
   ...
   qsort(lista,numeroelementos,sizeof(tipo),&nombrefun);
   ...
```

Ejemplo:

```
1  int intcom(const void *e1, const void *e2) {
2  const int *el1, *el2;
3  el1=(int *)e1;
4  el2=(int *)e2;
5  return (*el1)-(*el2);
6  }
7  ...
8  void main()
9     {
10    int cuantos;
11
12    printf("Cuantos: ");
13    scanf("%i",&cuantos);
14
15       {
16       int lista[cuantos];
17       /* Llamada para pedir la lista */
18      ...
19       /* Ordenar  */
20       qsort(lista,cuantos,sizeof(lista[0]),&intcom);
21      .....
```

Programación general

Hay distintos métodos, que pueden verse en bastantes libros. Las preferencias entre unos y otros son por eficiencia. Nosotros no entramos a tanto, y comentamos aquí dos: uno fácil de entender (conocido como método de selección directa) y otro bastante eficaz (método de Shell con incrementos de Sedgewick), útil para listas de valores de hasta decenas de millares.

Si trabajáramos con listas aún mayores, elegiríamos otros métodos aún más eficaces, pero también más complejos. Si la lista cabe en la memoria principal iríamos al método de ordenación rápida ("quicksort"). Si no cabe, iríamos a un método de ordenación por mezclas o intercalación. No vamos a explicar aquí estos métodos.

Método fácil de entender. Este método vale para listas pequeñas, porque aunque no sea muy rápido, en esos tamaños no se nota. Como es más fácil de entender, es más fácil de retocar y hacer alguna adaptación sin destrozarlo.

La idea es buscar el más pequeño e intercambiarlo con la primera posición, luego el siguiente más pequeño con la segunda, etc.

Plantilla/ejemplo:

```
1  void ordenar(int cuantos, int lista[cuantos]) {
2    int orden, min, posmin, resto;
3
4    for(orden=0;orden<cuantos-1;orden++) {
5    /*Este ciclo va recorriendo las posiciones en que se van
       dejando los menores que se encuentran */
```

```
6       min=lista[orden];
7       //Empezamos poniendo como menor el que nos toca
8       posmin=orden;
9       for(resto=orden+1;resto<cuantos;resto++) {
10      //Miramos los siguientes a ver si son menores que el
11        if (lista[resto]<min) {
12        //Si es menor, lo guardamos y anotamos su posicion
13          min=lista[resto];
14          posmin=resto;
15        }
16      }
17      lista[posmin]=lista[orden];
18      /*Intercambiamos el menor que hemos encontrado con
19      la posicion por la que ibamos */
20      lista[orden]=min;
21    }
22  }
23
24  void main() {
25    int cuantos, lista[500];
26
27    cargar(lista,&cuantos);
28    ordenar(cuantos,lista);
29    ...
30  }
```

Método eficaz para listas grandes. Se trata de ir comparando cada uno con otro posterior e intercambiarlos si no están ordenados. Con una pasada no se garantiza que queden toda la lista ordenada, porque no se han comparado todos con todos, sino sólo con uno distanciado. Entonces se repite con una distancia menor, hasta llegar a comparar con el siguiente. Si todos y cada uno son menores que el siguiente, entonces sí que están todos de menor a mayor.

Plantilla:

```
........
   void ordenar( int cuantos, tipo lista[tam]) {
   int
      incrementos[]={1,5,19,41,109,209,505,929,2161,3905,8929,16001,36289,64769};
   int indsalto, salto, i, orden;
      float temp;

      for (indsalto=13;indsalto>=0;--indsalto) {
       for (salto=incrementos[indsalto],orden=salto; orden<cuantos; orden++)
      {
         temp = lista[orden];
         for (i=orden ; i-salto >= 0 && temp < lista[i-salto]; i-=salto) {
               lista[i]=lista[i-salto];
```

```
            }
          lista[i] = temp;
          }
       }
   }



          ordenar(cuantosson, nuestralista);
```

Ejemplo:

```
1  ........
2  void ordenar( int cuantos, float lista[cuantos]) {
3  int incrementos
      []={1,5,19,41,109,209,505,929,2161,3905,8929,16001,36289,64769};

4  int indsalto, salto, i, orden;
5     float temp;
6
7     for (indsalto=13;indsalto>=0;--indsalto) {
8      for (salto=incrementos[indsalto],orden=salto; orden<cuantos;
           orden++) {
9        temp = lista[orden];
10       for (i=orden ; i-salto >= 0 && temp < lista[i-salto]; i-=
            salto) {
11              lista[i]=lista[i-salto];
12           }
13       lista[i] = temp;
14       }
15    }
16 }
17
18
19 void main()
20    {
21    int cuantos=0;
22
23    printf ("dame cuantos \t");
24    scanf("%i", &cuantos);
25       float lista[cuantos];
26
27       .........
28
29       /*ahora ordenamos */
30
31       ordenar(cuantos, lista);
```

```
32
33          ................
34      }
```

Ordenar una lista usando la macro suministrada

```
1  #include <stdio.h>
2  #include <listas.h>
3  #define MAX 100
4  void lee(int *canti, int lista[MAX])
5  /* Función que lee los datos
6  ENTRADA: ---------
7  SALIDA: cuantos son y la lista */
8     {
9     printf("Cuantos numeros en lista: ");
10    scanf("%i",canti);
11    printf("Teclearlos\n");
12    lelisnu("%i",lista,*canti);
13    }
14 void escribe(int canti, int lista[MAX])
15 /* Función que ordena y escribe
16 ENTRADA:  cuantos son y lista
17 SALIDA: ---------- */
18    {
19    ordena(lista,canti);
20    eslisnu("%i ",lista,canti);
21    }
22 void main()
23    {
24    int canti, lista[MAX];
25    lee(&canti, lista);
26    escribe(canti,lista);
27    }
```

Búsqueda

Es un caso de recorrido leyendo o por lista (ver plantilla correspondiente). El recorrido termina cuando cierto valor leído cumple cierta condición (es decir, que se sigue si cumple la contraria). Por ejemplo, para buscar un valor pedido sería:

```
1 printf("Valor a localizar: ")
2 scanf("%f",&x);
3 for(posicion=0;lista[posicion]!=x;posicion++);
```

Puede ser también que cumpla cualquier otra condición.

Buscar un valor en una lista

```
1  #include <stdio.h>
```

```
2  #include <listas.h>
3  void escribe(int canti, int lista[canti], int buscado)
4  /* Función que busca y escribe
5  ENTRADA:  cuantos son, lista y buscado
6  SALIDA: ---------- */
7     {
8     int posic;
9     for (posic = 0; posic < canti; posic++)
10       {
11       if (lista[posic] == buscado)
12          {
13          printf("Aparece en posicion %i\n", posic+1);
14          }
15       }
16    }
17 void main()
18    {
19    int canti, buscado;
20    printf("Numero a buscar: ");
21    scanf("%i",&buscado);
22    printf("Cuantos numeros en lista: ");
23    scanf("%i", &canti);
24       {
25       int lista[canti];
26       printf("Teclearlos\n");
27       lelisnu("%i",lista,canti);
28       escribe(canti,lista,buscado);
29       }
30    }
```

Como la lista no está ordenada, se utiliza una búsqueda lineal, recorriendo la lista.

Extremo

Es un caso de recorrido leyendo o por lista(ver plantilla anterior). A la variable en que se va a guardar el extremo hay que darle un valor inicial opuesto a lo que se busca (por ejemplo, un valor muy grande si se busca un mínimo). También se puede dar como valor inicial el primero y empezar a comparar a partir del segundo.

Plantilla:

```
valor inicial de extremo
   for(posicion=0; posicion<tamaño; posicion++) {
   if (lista[posicion] es más extremo) {
   extremo=lista[posicion];
   }
   }
```

Ejemplo:

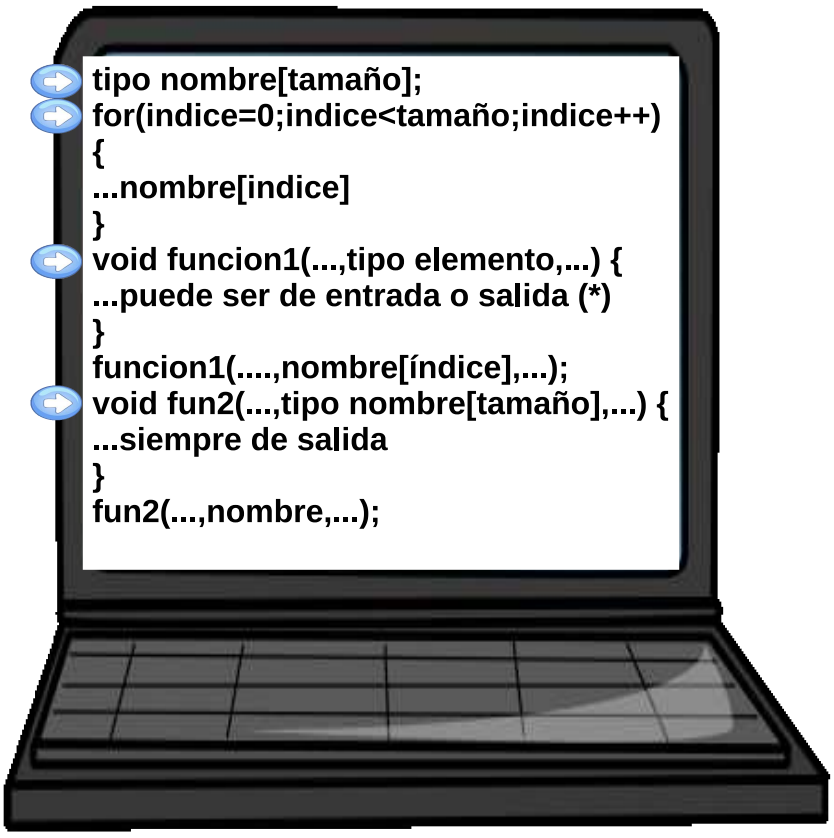
```
1 minimo=100000;
2 for(posicion=0; posicion <= cuantos; posicion++) {
3 if (lista[posicion]<minimo) {
4 minimo=lista[posicion];
5 }
6 }
```

Construcción de lista

Es un caso de recorrido leyendo o por lista (ver plantilla anterior). En este caso las instrucciones son dar valores a elementos de una lista.

Ejemplo:

```
1 for(posicion = 0; posicion < cuantos; posicion++) {
2 cuadrados[posicion]=valores[posicion]*valores[posicion];
3 }
```



Ejemplos de instrucciones

Aquí se presentan ejemplos de instrucciones de lo que hemos visto.

```
error=fun[nit]-y;
   Tm cargas[CAP];
   poten[n]=ten[n]*int[n];
   scanf("%i",&paginas[n]);
   void preguntar(Minutos llamadas[NUMLL], int *nll) {...}
   scanf("%i %f",carne[cper],estat[cper]);
   recoger(&carnes[0]);
   color[max-1]=0;
   color[0]=255;
   for(con=tot-1; con>=0; con--) {     printf("%f\n",veloc[con]);}
   Lumen fuentes[nluz];
   incpl=precip[m]-precip[m-1];
   printf("Extremos: %i,%i\n",saldo[0],saldo[n-1]);
   incpl=precip[m]-precip[m-1];
   printf("%f ",hora[cuen]);
   for(mes=0; mes<MANHO; mes++) {      scanf("%f",&cosec[mes]);}
```

Cuestiones de autoevaluación

Preguntas

1-Un programa tiene que calcular el día de la semana de una fecha del año cualquiera pedida por el usuario, sabiendo el día de la semana del 1 de enero. Necesita una lista con los ...
que tiene cada ... (dar las dos palabras que faltan separadas con un espacio)

2-Dar la definición de una lista de tamaño MAXIMO llamada dni donde se van a almacenar números de DNI. Poner sólo los espacios imprescindibles. En esta instrucción no se definirán más variables.

3-Un programa tiene que leer una serie de números y calcular su media. ¿Necesita cuántos ciclos en la parte del cálculo de la media? Contestar con el número

4-Una función puede dar como resultado una lista. ¿Siempre, sólo de tamaño constante, sólo de tamaño ajustado o nunca? Contestar la opción elegida como se ha escrito en la pregunta.

5-Para saber si un número tecleado por el usuario es mayor que todos los números de una lista que ya está ordenada de menor a mayor es imprescindible usar ¿cuántos ciclos? Dar la contestación numérica.

6-Una función recibe como dato un vector de tamaño adaptado a lo que pide el usuario. La definición se pone:
void funmover(float vector, int tamanho)
Volverla a poner corregida especificando el tamaño del vector. Usar sólo los espacios imprescindibles.

7-¿Cuántos tipos distintos de variables puede haber en una única lista? Dar el valor numérico.

8-Si hay que calcular la raíz cuadrada de la media de una lista de números hay que usar un ciclo para recorrer la lista. La instrucción con la raíz cuadrada se pondría ¿antes, en la cabecera, dentro o después del ciclo?. Contestar antes, cabecera, dentro ó despues.

9-Se quieren intercambiar los valores de las posiciones i y j en la lista a. Dar las instrucciones necesarias usando una variable auxiliar llamada temp para el valor de la posición i. Poner todas las instrucciones seguidas separadas por ;

10-Si la lista a tiene de tamaño TAMANHO y usamos la instrucción:
y=a[i+1];
¿Cuál es el valor máximo que puede tener i? No poner espacios

11-Se quieren desplazar los valores de la lista a, de tamaño cuantos, una posición a la izquierda, de forma que el 2º vaya a la 1ª posición, el 3º a la 2ª, etc. En la última posición se pondrá el primer valor. Para ello se usa un ciclo con la siguiente cabecera:
for(temp=a[0],ind=0;ind<cuantos-1;ind++)
Justo detrás del ciclo se pone la instrucción: a[cuantos-1]=temp;
¿Qué única instrucción va dentro del ciclo? No poner espacios.

12-Hay que hacer un programa que pida al usuario una serie de números y escriba la raíz cuadrada de su suma. ¿De cuál de las siguientes maneras se puede hacer?
a-Con una lista para los números
b-Sin ninguna lista, trabajando con una variable normal para cada número de la lista
Contestar a, b, ab, o ninguna (si ambas están mal)

13-Dar la instrucción C que pone en el primer elemento de una lista llamada lisej el segundo menos el tercer elementos. No poner espacios.

14-Se tiene una lista de números (llamada lisnum) y otra de letras (llamada lislet) que se corresponden uno a uno. Una parte previa del programa ha calculado el valor máximo de la lista de números y lo ha puesto en la variable max, y ha puesto en la variable pos su posición en la lista. Si se quiere escribir la letra correspondiente a ese

valor máximo, dar la instrucción C que lo hace, sin espacios y sin escribir salto de línea, sólo la letra.

15-Un producto escalar entre dos vectores es la suma de los productos entre los componentes correspondientes de cada uno. ¿Cuál es el mínimo número de ciclos para hacer esta operación?

16-Si se tienen tres listas independientes y se quiere comprobar si cierto valor está en alguna de ellas. ¿Cuántos ciclos hacen falta?

17-Se tienen dos listas lista1 y lista2 que están ordenadas de menor a mayor. La primera tiene total1 elementos y la segunda tiene total2. Se pone un condicional para ver cúal de las dos listas tiene el elemento mayor. Es:

```
if(lista1[___] > lista2[____])
```

Dar lo que va en cada hueco, cada uno sin espacios y separados el uno del otro por un espacio.

18-Se van a leer dos listas y se va a mirar el valor mínimo en cada posición entre las dos listas (el mínimo entre los primeros, el mínimo entre los segundos, etc.). ¿Puede hacerse esta operación usando sólo dos listas o hacen falta tres? Contestar con el número.

19-Se tienen dos listas, lista1 y lista2 y una posición pos. Se quieren intercambiar los valores de las listas en esa posición. La primera instrucción es

```
aux=lista1[pos];
```

Dar las otras dos, sin espacios y en la misma línea

20-Se tienen dos listas, lista1 y lista2 y una posición pos. Se quiere escribir la segunda lista insertada en la primera, en la posición pos. Se plantean las siguientes estrategias:

a-Ciclo para escribir la primera hasta la posición pos. Después, ciclo para escribir la segunda. Después, ciclo para escribir el resto de la primera.

b-Ciclo para escribir la primera. Dentro de él un condicional que compruebe si estamos en la posición pedida y entonces un ciclo para escribir la segunda.

Contestar a, b, ab o ninguna, según cuáles sean válidas

21-Se van a leer valores hasta que haya una cierta cantidad que sean cada uno distinto del siguiente. Otro caso distinto es que sea esa cantidad todos distintos entre sí. Simplemente se trata de verificar estas condiciones, sin hacer ningún cálculo ¿Hacen falta listas en estos dos casos? Contestar sí/no para cada caso, separados por un espacio.

22-Se tiene la lista lisnum que tiene de tamaño MAXPOSIB. ¿Qué hay que

poner dentro de los paréntesis de un if para comprobar si el elemento de la posición pos es mayor que el anterior? Ponerlo por el orden preguntado y sin espacios.

23-Se tiene una lista posic de tamaño cuan y se quiere formar otra con las diferencias entre valores consecutivos de esa lista. ¿Qué tamaño tendrá esta segunda lista? Y si se forma una tercera con las diferencias entre valores consecutivos de esta segunda, ¿qué tamaño tendrá? Dar las dos respuestas sin más espacio que el de separación entre ellas.

24-Se tiene una lista pot de valores con decimales, que se va a leer, pero el usuario no va a dar los valores desordenados, dando en cada caso posición (las cuenta desde 1) y valor. Para cargar la serie se hace un ciclo dentro del cual se lee primero posición y luego valor de la lista. La instrucción de lectura de posición es:

```
scanf("%i",&posic);
```

¿Cuál es la instrucción siguiente, de lectura del elemento correspondiente de la lista? No poner espacios.

25-Se tiene una lista lisnum y se quiere calcular las sumas de los sucesivos grupos de tres (1º2º3º, 4º5º6º, 7º8º9º, etc.). Para ello se plante un ciclo. Cabeceras alternativas:

a-for(pos=1;pos<lim;pos++)
b-for(pos=0;pos<lim-2;pos+=3)

Expresiones de suma alternativas para poner dentro del ciclo:

a-lisnum[pos]+lisnum[pos+1]+lisnum[pos+2]
b-lisnum[pos-1]+lisnum[pos]+lisnum[pos+1]

Dar todas las combinaciones válidas separadas por espacios. Por ejemplo aa ab ac ba bb bc, o n si no hay combinación válida

26-Dar la cabecera de una función que se llama leer, no recibe datos y da como resultado una lista de números con decimales llamada lisnum de tamaño TOTAL. No poner más espacios que los imprescindibles.

27-Un programa llama a una función mostrar pasándole la variable sin decimales cuant y recibiendo como resultado la lista jstar de números sin decimales, cuyo tamaño está en cuant. Escribir la llamada sin espacios, poniendo primero dato y luego resultado.

28-Un programa recorre una lista buscando la primera aparición de un número, que tiene que escribir. Para ello, dentro del ciclo se pone un if que compara el elemento correspondiente de la lista con el buscado. Dentro del if se pone el printf para escribir la posición en que se haya encontrado y ... ¿qué otra instrucción? Ponerla tal y como aparecerá en el programa.

29-Si en cierto momento un programa pone un valor en una posición de una lista, más allá de su tamaño, ¿qué pasará?
a-Siempre dará un error inmediato
b-Siempre se producirá error, pero puede ser inmediato o diferido
c-Puede que se produzca error y puede que no
d-El programa dará los resultados correctos en cualquier caso, pero ocupará más memoria
Dar la letra correcta, o n si no vale ninguna

30-Se quiere ver si dos listas que tienen el mismo tamaño son iguales, es decir, tienen los mismos números. Para ello se pone:
if (lista1==lista2)
¿Saldrá bien?
a-Sí
b-Depende del tipo de elementos que tengan las listas
c-Depende de si el tamaño de las listas es fijo o adaptado a lo que pide el usuario
Dar la letra correcta, seguidas y por el orden pedido si hay más de una correcta, o n si ninguna vale

31-Para probar un programa que trabaja con una serie de números se dispone de varios ejemplos resueltos, variando la longitud de la serie. ¿Cuál se elegiría para la primera prueba, si las longitudes son las siguientes?
a-3
b-15
c-8
d-90
Dar la letra elegida

32-Se tiene una lista y un número suelto y se quiere escribir la diferencia de cada elemento de la lista a ese número dividida por la mayor de ellas. ¿Cuántos ciclos hacen falta?

33-Un programa tiene que leer una serie de valores, para cada uno de los cuales calcular otro y escribir la suma de estos otros. Estrategias alternativas:
a-Ciclo de lectura en lista. Ciclo de cálculo en otra lista. Ciclo de suma. Escribir suma
b-Ciclo de lectura, cálculo y suma; sin usar listas. Escribir suma.
c-Ciclo de lectura y cálculo en lista. Ciclo de suma. Escribir suma.
Dar las opciones válidas juntas, o n si no vale ninguna

34-Se tienen dos listas de tamaño diferente y se quiere obtener el producto de todos los números de las dos listas. ¿Cuántos ciclos hacen falta?

35-Se tiene que indicar en qué piso quedan una serie de puntos de una fachada dada su altura desde el suelo. Para ello se dispone de la altura de los pisos del edificio. Indicar en cuáles de los siguientes casos hacen falta guardarlas en una lista.
a-Son todas iguales
b-Son todas diferentes
Indicar a,b,ab o n si en ninguno de los dos casos es necesario

Respuestas

1 días mes

2 int dni[MAXIMO];

3 1

4 Siempre

5 0

6 void funmover(int tamanho, float vector[tamanho])

7 1

8 Después

9 temp=a[i];a[i]=a[j];a[j]=temp;

10 TAMANHO-2

11 a[ind]=a[ind+1];

12 ab

13 lisej[0]=lisej[1]-lisej[2];

14 printf("%c",lislet[pos]);

15 1

16 3

17 total1-1 total2-1

18 2

```
19 lista1[pos]=lista2[pos];lista2[pos]=aux;

20 ab

21 no sí

22 lisnum[pos]>lisnum[pos-1]

23 cuan-1 cuan-2

24 scanf("%f",&pot[posic-1]);

25 ba

26 void leer(float lisnum[TOTAL])

27 mostrar(cuant,jstar);

28 break;

29 c

30 n

31 a

32 2

33 abc

34 2

35 b
```

Ejemplos resueltos: Listas

Sumas de números que cumplan condiciones

Vamos a plantear con listas dos problemas que hemos resuelto sin ellas:

> Calcular todas las sumas de números obtenidas empezando por uno dado, terminando en otro dado y separados un salto entero que varíe entre dos límites dados también por el usuario. Sólo se sumarán los que sean múltiplos de cierto valor entero dado.

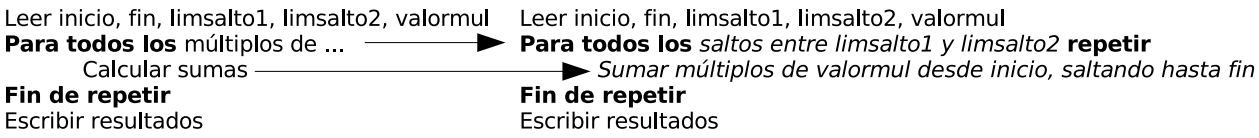
Resolución

Inicialmente planteamos unos bloques gordos basados en el propio enunciado:

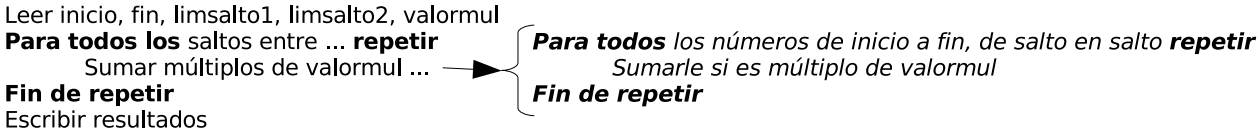
Leer inicio, fin, limsalto1, limsalto2, valormul
para todos los casos de múltiplos de valormul obtenidos empezando por inicio y saltando un valor que varíe entre limsalto1 y limsalto2, terminando en fin **repetir**
  Calcular sumas
final de **repetir**
Escribir resultados

Cada uno de esos tres bloques puede ser una función.

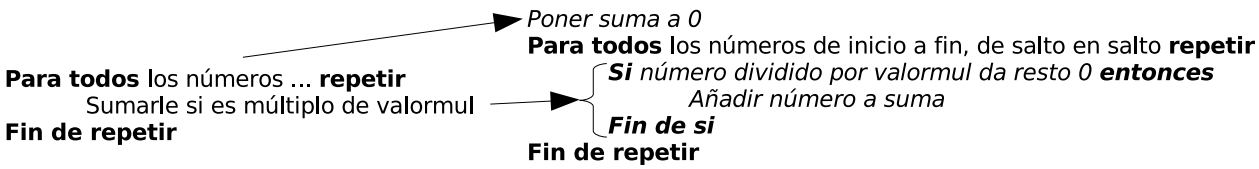
Vamos a detallar la fase de cálculo dentro de la idea de repetir el asignar un resultado a una variable; como dice todas las sumas que salgan variando el salto, ponemos un ciclo que haga esa variación. Esa fase habría que cambiarla:



La instrucción de cálculo que está dentro de la repetición hay que desglosarla todavía más; podría ser una función. Para detallarla volvemos a aplicar el planteamiento de repetición de cálculo, observando que la suma total es una repetición de sumas individuales. Este desarrollo sería:



Para la suma necesitaríamos una variable que habría que poner a 0 antes de empezar y para ver si es múltiplo comparamos su resto con 0. Para avanzarla así habría que cambiar:



Simplemente quedaría definir todas las variables y el paso entre las distintas funciones. Como hemos separado la escritura de resultados al final, hay que guardarlos todos hasta entonces, por lo que podemos hacerlo en una lista.

Medias

Dada una cantidad de valores, calcular por separado las medias de los que están por encima y por debajo de un número dado.

Resolución

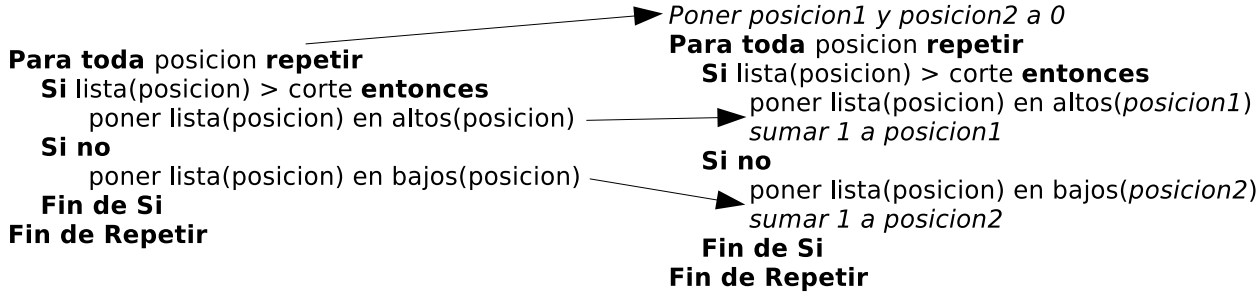
En este caso puede verse que una idea es separar los valores que están por encima de los que están por debajo, cada uno en sendas listas, para luego calcular la media de cada uno. El esquema global del programa sería:

Leer lista de valores y corte
Separar en dos listas de altos y bajos
Calcular media de altos
Escribirla
Calcular media de bajos
Escribirla

Para separar la lista original en dos, que puede ser una función separada, existe una plantilla para construir una lista, pero este caso requiere una modificación, ya que vamos a guardar por

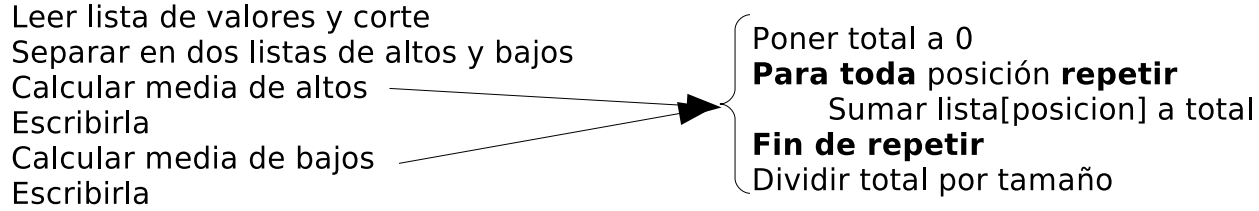
ejemplo los altos todos seguidos, sin huecos en medio, para procesarlos después directamente al hacer la media. La aplicación literal de la plantilla nos llevaría a:

Si queremos guardarlos todos seguidos, entonces sus posiciones no van a ser la originales, sino distintas, y entonces hay que guardarlas en variables distintas, que empezaran en 0 (pensando en el C, que es así) y se incrementarán cada vez que metamos uno. Quedaría:



Al final tendríamos en posicion1 y posicion2 el tamaño de las dos listas respectivamente.

El siguiente punto a desarrollar es el cálculo de media. El proceso es análogo para altos que para bajos, con lo que se puede usar la misma función, pasándole cada vez una lista diferente (con un número de valores diferente). La suma es una acumulación y para eso tenemos una plantilla. Por ser suma el valor inicial del acumulador es 0. Tras la acumulación de sumas habría que dividir por el número de elementos. La función quedaría:



Para pasarlo a C lo primero es dividirlo en funciones. Tenemos dos bloques que hemos desarrollado por separado: el de separar en dos listas y el de sacar la media de una lista, así que haremos esas funciones. Empecemos por el de separar en dos listas; traduciendo a C nos queda:

```
1 posicion1=0;
2 posicion2=0;
3 for( posicion=0;posicion<cuantos;posicion++) {
4         if (lista[posicion] > corte) {
5                 altos[posicion1]=lista[posicion];
6                 posicion1++;
7         } else {
8                 bajos[posicion2]= lista[posicion];
9                 posicion2++;
```

```
10          }
11  }
```

Veamos el tipo de cada variable y si son argumentos:

Variable	Tipo	Argumento
posicion1	Sin decimales, es un recuento	Se le da valor en la función y lo usamos después: argumento de salida
posicion2	Sin decimales	argumento de salida
posicion	Sin decimales, es una posición	Sólo se usa dentro de la función: variable local
cuantos	Sin decimales, es un recuento	Llega leída desde main, no se modifica en la función: argumento de entrada
lista	Conjunto de valores posiblemente con decimales. Si suponemos que su cantidad nos la dicen a priori, ése será su tamaño.	Llega leída desde main, no se modifica en la función: argumento de entrada (aunque en listas da igual que sea de entrada o de salida)
corte	Con decimales	Llega leída desde main, no se modifica en la función: argumento de entrada
altos	Conjunto de valores posiblemente con decimales. Como no sabemos de entrada cuántos serán, le daremos el mismo tamaño que a la lista completa.	Se le da valor en la función y lo usamos después: argumento de salida
bajos	Igual que el anterior	Igual que el anterior

Añadiendo entonces las declaraciones y la cabecera de la función ya la dejamos completa:

```
1  void separar(int *posicion1, int *posicion2, int cuantos, float
       lista[cuantos], float corte,
2           float altos[cuantos],float bajos[cuantos]) {
3  int posicion;
4
5  *posicion1=0;
6  *posicion2=0;
7  for( posicion=0;posicion<cuantos;posicion++) {
8           if (lista[posicion] > corte) {
9                   altos[posicion1]=lista[posicion];
10                  (*posicion1)++;
11          } else {
12                  bajos[posicion2]= lista[posicion];
13                  (*posicion2)++;
14          }
15 }
16 }
```

Observa que para evitar el conflicto entre el * de argumento de salida y el ++ hemos usado paréntesis.

Vamos ahora con la otra función, la de calcular la media de una lista. Empezamos traduciendo a C las instrucciones que habíamos deducido:

```
1 total=0;
2 for( posicion=0;posicion<tamano;posicion++) {
3         total+= lista[posicion];
4 }
5 media= total / tamano;
```

Observa que como las eñes no se permiten, hemos usado el nombre tamano.

Vamos ahora con las variables:

Variable	Tipo	Argumento
total	Con decimales, porque es una suma de números con decimales	Sólo se usa dentro de la función: variable local
posicion	Sin decimales, es una posición	Sólo se usa dentro de la función: variable local
tamano	Sin decimales, es un recuento	Llega desde main, no se modifica en la función: argumento de entrada
lista	Conjunto de valores posiblemente con decimales. El tamaño es la variable anterior.	Llega desde main, no se modifica en la función: argumento de entrada (aunque en listas da igual que sea de entrada o de salida)
media	Con decimales	Se le da valor en la función y lo usamos después: argumento de salida

Poniendo las declaraciones y cabecera acordemente con lo anterior tenemos:

```
1  void calmedia(int tamano, float lista[tamano], float *media) {
2  float total;
3  int posicion;
4
5  total=0;
6  for( posicion=0;posicion<tamano;posicion++) {
7          total+= lista[posicion];
8  }
9  *media= total / tamano;
10 }
```

Nos queda sólo `main`. Ahí tenemos la lectura de datos, para la que vamos a suponer que están en el fichero "medidas.txt", llamadas a las funciones anteriores y escritura de resultados. Metemos ya las variables, puesto que las hemos tratado en las funciones. Sólo tenemos que tener la precaución de que las declaraciones de listas de tamaño variable van después de haber cargado esa variable.

En las llamadas no es obligatorio que se usen los mismos nombres, pero sí la posición.

```
1 void main() {
2 FILE *fichero;
3 int cuantos,posic,cuantos1,cuantos2;
4 float corte,media1,media2;
```

```
5
6  fichero=fopen("datos.txt","r");
7  fscanf(fichero," %i",&cuantos);
8  float lista[cuantos],altos[cuantos],bajos[cuantos];
9  fscanf(fichero," %f",&corte);
10 for(posic=0;posic<cuantos;posic++) {
11         fscanf(fichero," %f",&lista[posic]);
12 }
13 separar(&cuantos1, &cuantos2, cuantos, lista, corte,
14         altos, bajos);
15 calmedia(cuantos1, altos, &media1);
16 calmedia(cuantos2, bajos, &media2);
17 printf("La media de los altos es %f\n",media1);
18 printf("La media de los bajos es %f\n",media2);
19 }
```

El programa completo se obtiene simplemente poniendo el `include` seguido de las funciones por el orden que las hemos desarrollado.

Operaciones elementales con listas

```
1  #include <stdio.h>
2  #define TAM_VECTOR 10
3  void main() {
4   int vector_a[TAM_VECTOR];
5   int vector_b[TAM_VECTOR];
6   int j; /* variable utilizada como indice */
7
8  /* leer el vector a */
9  for (j = 0; j < TAM_VECTOR; j++) {
10    printf("Elemento %d: ", j);
11    scanf("%d", &vector_a[j]);
12 }
13
14 /* copiar el vector */
15 for (j = 0; j < TAM_VECTOR; j++) {
16     vector_b[j] = vector_a[j];
17 }
18
19 /* escribir el vector b */
20 for (j = 0; j < TAM_VECTOR; j++) {
21     printf("El elemento %d es %d \n", j, vector_b[j]);
22 }
23 }
```

Ordenación y búsqueda

```
1  /* Implementacion del algoritmo quick-sort para la ordenacion de
2  vectores  */
```

```
3
4  #include <stdlib.h>
5  #include <stdio.h>
6  #include <time.h>
7
8  #define ELEMENTOS 100
9
10 /* Funcion de comparar */
11 int comparar(const void *arg1, const void *arg2)
12 {
13  if(*(int *)arg1 < *(int *)arg2) return -1;
14  else if(*(int *)arg1 > *(int *)arg2) return 1;
15  else return 0;
16 }
17
18 void main()
19 {
20    int i, num;
21    int lista[ELEMENTOS], int *elementoPtr;
22
23    /* Contenido aleatorio entre 1 y 100 */
24         srand(time(NULL));
25    for(i = 0; i < ELEMENTOS; i++) {
26         lista[i] = rand() % 100 + 1;
27    }
28
29    /* Quick-Sort */
30    qsort(lista, ELEMENTOS, sizeof(lista[0]), comparar);
31
32     /* Mostramos la lista ordenada */
33    for(i = 0; i < ELEMENTOS; i++) {printf("%i ", lista[i]);}
34    printf("\n");
35
36
37 }
```

En este caso se ha utilizado la función incluida en el lenguaje C.

Leer una lista de valores y escribirla en orden inverso

```
1  #include <stdio.h>
2
3  void lee(int cantidad, int lista[cantidad])
4     {
5     /* Para llevar la cuenta de los que se leen */
6     int conta;
7     for (conta = 0; conta < cantidad; conta++)
8        {
```

```
9         /* El numero de orden se saca empezando en 1 */
10        printf("Valor %i: ", conta+1);
11        scanf("%i", &lista[conta]);
12        }
13    }
14
15 void escribe(int cantidad, int lista[cantidad])
16    {
17    /* Para llevar la cuenta de los que se escriben */
18    int conta;
19    for (conta = cantidad-1; conta >= 0; conta--
20          /* Como se escriben del reves,
21          empieza por el ultimo y retrocede */)
22        {
23        printf(" %i", lista[conta]);
24        }
25    }
26
27 void main()
28    {
29    int cantidad;
30    printf("Cuantos son?");
31    scanf("%i", &cantidad);
32        {
33        int lista[cantidad];
34        lee(cantidad,lista);
35        escribe(cantidad,lista);
36        }
37    }
```

En el ciclo de escritura de salida se sigue una cuenta atrás de posiciones.

Calcular tiempos que tardan trenes en recorrer tramos

```
1  /* ENUNCIADO
2  En un sistema ferroviario hay una serie de lÝneas que hay que
      servir y se dispone de una serie de trenes. Nos interesa
      calcular el tiempo que tardarß cada tren en recorrer su lÝnea
      . Se supone que los trenes van a velocidad uniforme, excepto
      en el arranque y en la llegada (en este caso no hay paradas
      intermedias). Para tener en cuenta el tiempo que se tarda de
      mßs por los arranques y llegadas, en cada lÝnea se leerß un
      tiempo a sumar por el arranque y el de llegada se supondrß
      que es igual. Hacer un programa que lea los datos de lÝneas (
      longitud y tiempo extra de maniobra en estaci¾n) y de trenes
      (velocidad y lÝnea que le toca) y escriba el tiempo que
      tardarß cada uno en hacer su lÝnea.
3
```

```
4  ESQUEMA DE SOLUCIËN
5  Pedir y leer cantidad de lÝneas/tramos ferroviarios
6  Leer la longitud de cada tramo en una lista ------- usar una
      funci¾n
7  Leer el incremento de tiempo por arranque/parada en cada tramo
      -------- usar funci¾n
8  Pedir y leer cantidad de trenes
9  Repetir, contando trenes
10            Pedir y leer su velocidad y la lÝnea/tramo que le toca
11            Calcular el tiempo (longitud del tal tramo/velocidad del
                  tren + 2*tiempo extra)
12            Escribirlo
13 */
14 #include <stdio.h>
15 typedef float Km, Kmh, Seg;
16 #define SEGHORA 3600
17 /*~TypeCombination
18 Km / Kmh -> Seg
19 */
20
21 void leelon(int ntramos, Km longit[ntramos])
22    {
23    int cuenta;
24    for (cuenta = 0; cuenta < ntramos; cuenta++)
25       {
26       scanf("%g",&longit[cuenta]);
27       }
28    }
29
30 void leetiemex(int ntramos, Seg tiemex[ntramos])
31    {
32    int cuenta;
33    for (cuenta = 0; cuenta < ntramos; cuenta++)
34       {
35       scanf("%g",&tiemex[cuenta]);
36       }
37    }
38
39 void main()
40    {
41    int ntramos, ntrenes, cuentren,tramo;
42    Kmh vel;
43    Seg tiempo;
44    printf("¿Cuántos tramos? ");
45    scanf("%i",&ntramos);
46       {
47       Km longit[ntramos];
```

```
48        Seg tiemex[ntramos];
49        printf("Dar longitudes de tramos\n");
50        leelon(ntramos,longit);
51        printf("Dar tiempos de arranque\n");
52        leetiemex(ntramos,tiemex);
53        printf("¿Cuántos trenes? ");
54        scanf("%i",&ntrenes);
55        for (cuentren = 0; cuentren < ntrenes; cuentren++)
56           {
57           printf("Velocidad de tren %i: ", cuentren+1);
58           scanf("%g",&vel);
59           printf("Tramo: ");
60           scanf("%i",&tramo);
61           tiempo = longit[tramo-1]/vel*SEGHORA + 2*tiemex[tramo
                -1];
62           printf("Tardará %g seg.\n",tiempo);
63           }
64        }
65     }
```

Las longitudes y tiempos extra a tener en cuenta se localizan en sus respectivas listas porque el usuario da la posición. Como se supone que el usuario empieza a contar en 1 y en C se empieza en 0, hay que restarle 1 al valor leído.

Escribir diferencias con el mínimo y si son múltiplos de él

```
1  #include <stdio.h>
2  #include <listas.h>
3  #define MAXLISTA 50
4  void escribe(int canti, int lista[MAXLISTA])
5  /* Función que calcula y escribe
6  ENTRADA:  cuantos son y lista
7  SALIDA: ---------- */
8     {
9     int posic, min;
10    min = lista[0];
11    for (posic = 1; posic < canti; posic++)
12       {
13       if (lista[posic] < min)
14          {
15          min = lista[posic];
16          }
17       }
18    printf("Diferencias con el minimo y si son multiplos:\n");
19    for (posic = 0; posic < canti; posic++)
20       {
21       printf("%i ",lista[posic]-min);
22       if (lista[posic]%min == 0)
```

```
23          {
24          printf("si\n");
25          }
26       else
27          {
28          printf("no\n");
29          }
30       }
31    }
32 void main()
33    {
34    int canti, lista[MAXLISTA];
35    printf("Cuantos numeros en lista: ");
36    scanf("%i", &canti);
37    printf("Teclearlos\n");
38    lelisnu("%i",lista,canti);
39    escribe(canti,lista);
40    }
```

Se utiliza la plantilla de buscar extremo y luego se recorre la lista.

Pasar de latitudes y longitudes a coordenadas en proyección gnomónica

```
1  /* ENUNCIADO
2  Se va a realizar un mapa de un paÝs utilizando la proyecci¾n
      gnom¾nica. Hacer un programa que lea las latitudes y
      longitudes de los puntos del contorno del paÝs y escriba las
      coordenadas de los puntos correspondientes en el mapa. El
      programa preguntarß inicialmente la cantidad de puntos, leerß
       la latitud y longitud de cada uno y escribirß finalmente las
       coordenadas en el mapa, en centÝmetros. El mapa estarß a
      escala 1/1000000
3  Las f¾rmulas de la proyecci¾n gnom¾nica son:
4  x= R tan(lo)
5  y= R tan(la)/cos(lo)
6  donde la es la latitud y lo es la longitud. Tomar como radio
      aproximado de la Tierra (R) un valor de 6400 Km
7
8  ESQUEMA DE SOLUCIËN
9  Pedir y leer cantidad de puntos
10 Leer latitudes y longitudes en sendas listas ------ usar funci¾n
11 ----------Funci¾n para calcular las coordenadas gnom¾nicas
12 Repetir contando puntos hasta llegar a la cantidad que son
13          Calcular coordenadas con las f¾rmulas
14          Escribirlas
15 */
16 #include <stdio.h>
17 #include <math.h>
```

```
18 #define CONVRAD (3.1416/180)
19 #define R 640000000
20 #define ESCALA 1000000
21 typedef float Grados, Cm;
22
23 void leell(int npuntos, Grados lat[npuntos], Grados lon[npuntos
      ])
24    {
25    int cuenta;
26    for (cuenta = 0; cuenta < npuntos; cuenta++)
27       {
28       scanf("%g %g",&lat[cuenta],&lon[cuenta]);
29       }
30    }
31
32 void cales(int npuntos, Grados lat[npuntos], Grados lon[npuntos
      ])
33    {
34    int cuenta;
35    Cm x,y;
36
37    for (cuenta = 0; cuenta < npuntos; cuenta++)
38       {
39       x = R*tan(lon[cuenta]*CONVRAD)/ESCALA;
40       y = R*tan(lat[cuenta]*CONVRAD)/cos(lon[cuenta]*CONVRAD)/
            ESCALA;
41       printf("Punto %i: x=%g y=%g\n",cuenta+1,x,y);
42       }
43    }
44
45 void main()
46    {
47    int npuntos;
48    printf("¿Cuántos puntos? ");
49    scanf("%i",&npuntos);
50       {
51       Grados lat[npuntos], lon[npuntos];
52       printf("Dar latitud y longitud de todos los puntos\n");
53       leell(npuntos,lat,lon);
54       cales(npuntos,lat,lon);
55       }
56    }
```

El cálculo se realiza con un recorrido de las listas, ya que es la misma posición en cada una.

Unión e intersección de listas

```
1 /* ENUNCIADO
```

```
2  Hacer un programa que lea dos listas de valores (numeros sin
      decimales) y, considerandolos como conjuntos, escriba el
      conjunto union y el conjunto interseccion de ambos. Puede
      suponerse que la primera es la mas pequeña.
3  ESQUEMA DE SOLUCION
4  Pedir y leer la cantidad de valores en cada conjunto
5  Pedir y leer los valores de cada conjunto
6  ---------------Funcion para calcular la union y la interseccion
7  Copiar el primer conjunto en la union
8  Poner como cantidad en la union la del primer conjunto, y la de
      la interseccion darle un valor inicial de 0
9  Recorrer los elementos del segundo conjunto
10         ------------Funcion para buscar el elemento
              correspondiente en el primer conjunto
11         Poner la posicion en que se ha encontrado a -1
12         Recorrer el primer conjunto
13                 Si el elemento correspondiente coincide con el
                      que se esta buscando
14                       La posicion en que se encuentra es esta
15                       Terminar
16         ------------------------------------------------------------

17         Si no se ha encontrado (la posicion ha quedado en -1)
18                 Subir 1 el tamaño de la union
19                 Poner este elemento en la ultima posicion
20         En otro caso (si se ha encontrado, la posicion es alguna
               real)
21                 Subir 1 el tamaño de la interseccion
22                 Poner este elemento en la ultima posicion
23 ------------------------------------------------------------------
24 Escribir la uni¾n
25 Escribir la intersecci¾n
26 */
27 #include <stdio.h>
28 #include <listas.h>
29 #include <string.h>
30 void busca(int canti, int lista[canti], int buscado, int *donde)
31 /* Función que busca numero en lista, sólo primera vez
32 ENTRADA:  cuantos son, lista y buscado
33 SALIDA: primera posición ó -1 si no está */
34    {
35    int posic;
36    *donde = -1;
37    for (posic = 0; posic < canti; posic++)
38       {
39       if (lista[posic] == buscado)
40          {
```

```
41          *donde = posic;
42          return;
43          }
44       }
45    }
46 void calcula(int canti1, int canti2, int lista1[canti1], int
      lista2[canti2],
47 int listun[canti1+canti2], int listin[canti1], int *cantiun, int
       *cantiin)
48 /* Función que calcula unión e intersección
49 ENTRADA: cantidad de las dos listas y las dos listas
50 SALIDA: unión, intersección y cantidad de cada una */
51    {
52    int posic, primpos;
53    memcpy(listun,lista1,canti1*sizeof(int));
54    *cantiun=canti1;
55    *cantiin=0;
56    for (posic = 0; posic < canti2; posic++)
57       {
58       busca(canti1,lista1,lista2[posic],&primpos);
59       if (primpos < 0)
60          {
61          (*cantiun)++;
62          listun[*cantiun - 1]=lista2[posic];
63          }
64       else
65          {
66          (*cantiin)++;
67          listin[*cantiin - 1]=lista2[posic];
68          }
69       }
70    }
71 void main()
72    {
73    int canti1, canti2, cantiun, cantiin;
74    printf("Cuantos en cada lista: ");
75    scanf("%i %i",&canti1,&canti2);
76       {
77       int lista1[canti1], lista2[canti2], listun[canti1+canti2],
             listin[canti1];
78       printf("Primera lista: ");
79       lelisnu("%i",lista1,canti1);
80       printf("Segunda lista: ");
81       lelisnu("%i",lista2,canti2);
82       calcula(canti1,canti2,lista1,lista2,listun,listin,&cantiun
            ,&cantiin);
83       printf("Union:\n");
```

```
84        eslisnu("%i ",listun,cantiun);
85        printf("\nInterseccion:\n");
86        eslisnu("%i ",listin,cantiin);
87        }
88    }
```

Para la unión se copia la primera lista y se recorre la segunda, añadiendo los que no están, incrementando en su caso la cantidad de valores que hay en la unión. Si ya están entonces se añaden a la de intersección, que ha empezado vacía.

Temperatura media

En la figura 0.13 tienes un programa que lee una serie de fechas (sólo el día) y la temperatura que ha hecho ese día, y escribe la temperatura media. Realmente este programa puede hacerse sin listas, pero así se ve su uso. Se presenta como diagrama de cajas, pero está todo el C prácticamente desarrollado y acabar de hacerlo es trivial.

Leer días con temperatura y escribir media

```
#include <stdio.h>
typedef int Dias;
typedef float Grados;
```

Primero cuántos son

La lista de días

La lista de temperaturas

```
function void leer(int cuantos, Dias listadias[cuantos], Grados temp[cuantos])
        /* Función que lee los días y sus temperaturas */
    int cuenta;
```

Va leyendo desde el 0 hasta el cuantos-1

```
    for ( cuenta = 0; cuenta < cuantos; cuenta++ )
        do /* Para que aparezcan empezando en 1, suma 1 a la cuenta*/
           printf("Caso %i. Día y temperatura:", cuenta+1);
        while ( scanf("%i %g",&listadias[cuenta],&temp[cuenta]) < 2
                /* Repite mientras no lea dos valores */              );
```

La lista de temperaturas

Primero cuántos son

```
function Grados calcular(int cuantos, Grados temp[cuantos])
        /* Función que calcula la media de las temperaturas */
    int cuenta;
    Grados suma;
    for ( suma=0,cuenta = 0; cuenta < cuantos; cuenta++
          /* La suma debe ponerse a 0 al empezar */      )
        suma += temp[cuenta];
```

Acumula la suma de temperaturas

```
    return suma/cuantos
           /* La media es la suma total dividido por cuantos son */ ;

function void main()
    int cuantos;
    Grados media;
    do printf("Cuantos son: ");
    while ( scanf("%i",&cuantos) < 1
            /* Repite la petición hasta que la lectura sea correcta */ );
    /*  Operación con listas variables  */
        Dias listadias[cuantos];
        Grados temp[cuantos];
        leer(cuantos,listadias, temp);
        media=calcular(cuantos,temp);
        /* El valor calculado se mete en media */
```

Primero lee y luego calcula la media

```
    printf("Temperatura media: %g\n",media);
```

Aquí ya no necesita las listas

¿Cómo se trabaja con textos?

¿Qué funciones de trabajo de texto tenemos?

C: Textos

Recursos en el aula virtual de textos

Teoría

Aparte de estos apuntes

Ejercicios iniciales de instrucciones concretas

Para comprobar que has interiorizado la teoría
Y recuerda las cuestiones de autoevaluación que tienes aquí en el apartado III

Ejemplos explicados

Con la solución y explicaciones
Y también en estos apuntes, en el apartado III

Ejemplos activos

Visualizaciones de la ejecución paso a paso de un programa

Ejemplos con solución

La solución sin o con pocas explicaciones
Y también en estos apuntes, en el apartado III

Ejercicios con pistas

Sin memorización de por dónde vas

Ejercicios propuestos

Sin la solución

Controles y exámenes de otros años

Algunos con solución, como ejemplos resueltos, y otros sin ella, como ejercicios propuestos

Un texto, en general, es un conjunto de caracteres. De hecho, pocas veces se usan los caracteres de uno en uno. Hablamos de caracteres y no de letras, porque las cifras, los signos de puntuación, los espacios, también pueden estar en los textos, también son caracteres.

Un texto o "cadena de caracteres" es un caso particular de lista. Se declara:

```
char nombre[num_carac];
```

y permite almacenar `num_carac-1` caracteres y el carácter nulo `'\0'` de terminación. Por ejemplo la variable `char frase[21];` es apta para almacenar 20 caracteres y el nulo.

C permite, además, la inicialización de cadenas de caracteres en la declaración, mediante sentencias del tipo

```
char cadena[ ] = "Esto es una cadena de caracteres";
```

en la que no es necesario añadir el nulo final ni indicar el tamaño, pues lo hace automáticamente el compilador. Otra forma de asignar un valor a una cadena es utilizando la función `strcpy` de la biblioteca `string.h`:

```
char cadena[10];
   strcpy(cadena, "Hola");
```

No se puede hacer: ~~`cadena="Hola";`~~

A la hora de lecturas y escrituras, C permite un tratamiento especial, que es la lectura/escritura de todos los caracteres con un sólo código de formato, pasando el texto entero. En el caso de escritura el código de formato es `%s`. En el caso de lectura se pueden usar:

- `%s` parándose al llegar un espacio
- `%[...]` parándose al llegar a un carácter que no esté entre los corchetes (pueden ser rangos, indicados con un guión)
- `%[^...]` parándose al llegar a uno de los carácteres que esté entre los corchetes tras el circunflejo; también pueden ser rangos

Un caso habitual de la última posibilidad es **cuando se quiere leer una frase completa, incluyendo espacios, hasta el final de línea: se usa** `" %[^\n]"`; el espacio primero es para pasar por encima de saltos de línea que hubiese y luego lee hasta el salto de línea.

Funciones para textos (string.h)

En todas estas funciones si en vez de usar `texto` desde el principio queremos usarlo o procesarlo a partir de la posición p, lo que hay que poner es `texto+p`

Función sprintf

Esta función actúa de igual modo que `printf` pero la salida es escrita en el texto `cadena`.
Esquema:

sprintf(cadena,formato, arg1,. . . , argN);

Ejemplo:

sprintf(cadena,” %f”, var);

Función sscanf

Esta función es equivalente a `scanf` y `fscanf` excepto que los caracteres de entrada son leídos de `cadena`.
Esquema:

sscanf(cadena,formato, arg1,. . . , argN);

Ejemplo:

sscanf(cadena,” %i”, &var);

Función strcat

Añade una copia de `s2` (incluyendo el carácter nulo) al final de `s1`. El carácter inicial de `s2` sobrescribe el carácter nulo al final de `s1`.

Ejemplo:

strcat(s1, s2);

Hay una variante para añadir sólo una parte del texto. Se llama `strncat` y tiene un tercer argumento que es la cantidad de letras o caracteres a añadir, por ejemplo:

strncat(s1, s2,n);

Función strcmp

Compara el texto `s1` con el `s2`. La función devuelve un número entero mayor, igual, o menor que cero, en función de si `s1` es mayor, igual, o menor, alfabéticamente, que `s2`.

Esquema:

strcmp(s1, s2)

Ejemplo:

```
1  #include <stdio.h>
2  #include <string.h>
3
4  void main() {
5     char s1[6] = "Abeja";
6     char s2[6] = "abeja";
7     int i;
8
9     printf( "s1=%s\t", s1 );
10    printf( "s2=%s\n", s2 );
11
12    printf( "s1 esta " );
13    if( strcmp( s1, s2 ) < 0 )  printf( "antes que" );
14    else if( strcmp( s1, s2 ) > 0 )  printf( "despues de" );
15    else  printf( "igual que" );
16    printf( " s2\n" );
17
18 }
```

Función strcpy

Copia `s2` (incluyendo el carácter nulo) a `s1`.

Ejemplo:

strcpy(s1, s2);

Hay una variante para copiar sólo una parte del texto. Se llama `strncpy` y tiene un tercer argumento que es la cantidad de letras o caracteres a copiar, por ejemplo:

strncpy(s1, s2,n);

No pone el carácter nulo al final

Función strlen

Calcula el número de caracteres de `s`, excluyendo el carácter nulo.
ejemplo:

numlet = strlen(s);

Función strncat

Añade n caracteres de `s2` al final de `s1`. El carácter inicial de `s2` sobrescribe el carácter nulo al final de `s1`. El carácter nulo siempre se añade al resultado.
Ejemplo:

strncat(s1, s2, n);

Función strncmp

Compara n caracteres de `s1` con `s2` y devuelve un número entero mayor, igual, o menor que cero según si `s1` es mayor, igual, o menor, alfabéticamente, que `s2`.
Ejemplo:

if (**strncmp(s1, s2, n)**==0) {

Función strncpy

Copia n caracteres de `s2` a `s1`.
Ejemplo:

strncpy(s1, s2, n);

Si `s2` es una cadena con menos de n caracteres entonces se añaden caracteres nulos hasta completar los n caracteres.

Función atof

La función convierte el texto `numPtr` a número fraccionario.
Ejemplo:

numerodec = atof(numPtr);

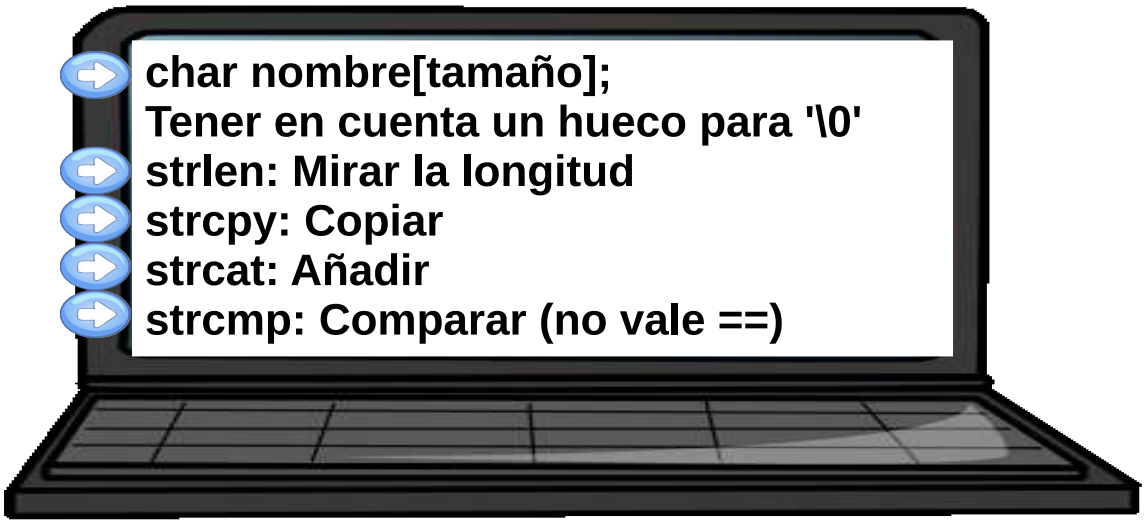
Función atoi

La función convierte el texto `numPtr` a entero.
Ejemplo:

numsindec = atoi(numPtr);

Ejemplo de uso:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main()
5  {
6      char numPtr[5] = "1234";
7
8      printf( "Convirtiendo la cadena \"%s\" en un numero: %d\n",
9       numPtr, atoi(numPtr) );
10
11 }
```



Ejemplos de instrucciones

Aquí se presentan ejemplos de instrucciones de lo que hemos visto.

```
printf("%c",palabra[cl]);
   printf("%s\n",respuesta);
   scanf(" %[^\n]",justif);
   scanf("%s",clave);
   char causa[LONFRA];
```

Cuestiones de autoevaluación

Preguntas

1-En un palabra aparecen (no seguidas) estas dos líneas:
#define MAXLETRAS 20
char nombre[MAXLETRAS]
¿Cuántas letras puede tener como máximo una palabra que vaya a almacenarse en la variable nombre?

2-Las palabras en las funciones pueden ser:
a-datos
b-resultados
Contestar a,b,ab o n (si no es ninguna)

3-Para leer una palabra que sabemos que va a ser de 4 letras una forma es:
scanf("%s",palabra);
La otra es leer las letras por separado. Dar la instruccin scanf correspondiente a esta otra forma, sin espacios, si las letras van a ir a las variables l1, l2, l3 y l4.

4-Poner sin espacios la instrucción que pone la longitud de la palabra "dato" en la variable "tamanho".

5-Dar las instrucción printf que escribe sólo la segunda letra del texto que está en la variable palabra, sin salto de línea posterior. No poner espacios.

6-En un programa tenemos las instrucciones (no consecutivas)
#define MAXLETRAS 20
char palabra[MAXLETRAS];
Cuando se lee de teclado la palabra correspondiente, se quiere estar seguro de que no se sale del espacio preparado. Se pone:
scanf("%__s",palabra);
¿Qué va en el hueco?

7-¿Qué función C sirve para buscar un texto dentro de otro?. Dar sólo su nombre.

8-¿Qué función C sirve para copiar un texto en otro, pero no hasta el final como strcpy, sino sólo una cierta cantidad de letras?. Dar sólo su nombre.

9-Un programa tiene que mirar si la letra de la posición pos en la variable texpal coincide con la variable let. El valor de pos lo ha dado el usuario contando desde 1. El if correspondiente es: if(_____ == let)
Dar lo que falta

10-¿Qué función C permite leer variables de una variable texto de forma análoga a como se lee de teclado?

11-Si hemos cargado en la variable texto el valor "ejemplo" y ponemos
printf("%s",&palabra[1]);
En pantalla sale jemplo
¿Qué sale si ponemos printf("%s",&palabra[0]);

12-Se tiene un texto en la variable palabra y su tamaño en la variable tam. Se quiere intercambiar la primera letra con la última. La primera instrucción que se usa es:
aux=palabra[0];
Dar las otras dos, sin espacios y en la misma línea

13-Se tiene un texto en la variable palabra y su tamaño en la variable tam. Se quiere cambiar la última letra poniendo el valor de la variable letra.
Dar la instrucción sin espacios

14-Dar un la condición que va dentro de un if para comprobar si la primera letra de la palabra pal es LETRA. No poner espacios.

15-¿Qué tamaño tiene que tener una variable donde se quieren almacenar 80 letras? ¿Y qué tamaño tiene que tener una variable donde se quieren almacenar textos (quizá con espacios) de hasta 80 letras máximo incluido? Dar los dos números separados por un espacio.

16-Se tienen la instrucción:
scanf("%[^\n]%c",palabra,&letra);
Cuando se ejecute, ¿qué valor tendrá letra? Ponerlo al estilo C

17-Se pone la instrucción siguiente:
if (scanf("%[^\n]",&texto)==__)
Queremos que al ejecutar, si el usuario teclea una línea en blanco (da directamente a aceptar sin teclear ninguna letra) la condición sea cierta. ¿Qué hay que poner en el hueco?

18-¿Qué función C compara dos textos?

19-Para usar las funciones de C que trabajan con textos hay que poner arriba
#include<____.h>
Rellenar.

20-Se tiene una palabra y se quiere escribir de ella sólo las letras que estén en una lista de permitidas, conservando el orden en que están en

la palabra. Estrategias alternativas:
a-Ciclo para recorrer la lista de permitidas y comprobar cada letra si está en la palabra. Si está, escribirla.
b-Ciclo para recorrer la palabra y comprobar cada letra si está en la lista de permitidas. Si está, escribirla.
Dar a, b, ab o n (para el caso de ninguna)

21-La función tolower pasa una letra a minúsculas. ¿Cuál es la función que la pasa a mayúsculas?

22-Si la letra let es mayúscula, la siguiente comparación dará verdadero:
if (isupper(let))
¿Por qué hay que cambiar isupper para que sea verdadero en caso de ser minúscula?

23-Un programa lee una frase incluyendo espacios en blanco en la variable tex. Para localizar palabras dentro de esta frase la va recorriendo, contando posiciones en la variable poslet, buscando espacios que vayan seguidos de algo que no sea espacio. Escribir el condicional en que aparezcan las comprobaciones por el orden presentado, agrupada cada una con un paréntesis y sólo con los espacios imprescindibles.

24-Un ciclo recorre una variable texto llamada cfras, contando las posiciones en la variable alet. Dentro del ciclo sólo hay un condicional: if ((alet%2)==0)
Este condicional sólo tiene dentro una instrucción, que está en el caso afirmativo: printf("%c%c",cfras[alet],cfras[alet]);
Si en cfras se introduce el valor "cpal" (comillas excluidas). ¿Qué saldrá en pantalla?

25-Una variable texto se llama pal y hay una posición en let (contada a partir de 1). La longitud de la palabra en pal está en long. Se quiere escribir la letra de la posición simétrica de let, es decir que si let es la primera posición, hay que escribir la última letra, si let es la segunda posición será la penúltima letra, etc. La instrucción es:
printf("%c",______);
Poner lo que va en el hueco, sin espacios.

26--Un programa lee una palabra en la variable tex. Interesa localizar las posiciones de la palabra donde aparece la letra let1 seguida de la letra let2. Para ello la recorre con un ciclo que usa la variable poslet. Escribir el condicional en que aparezcan las comprobaciones por el orden presentado, agrupada cada una con un paréntesis y sólo con los espacios imprescindibles.

27-En cierto punto de un programa queremos que la palabra varpal pase a tener el valor "clave" (comillas excluidas). Instrucciones

alternativas:
a) varpal="clave";
b) varpal='clave';
c) strcpy(varpal,"clave");
d) strcpy='clave';
Marcar la(s) correcta(s) sin separación o n si ninguna vale

28-Se quiere contar las coincidencias de letras en la misma posición entre dos palabras pal1 y pal2. ¿Cuántas variables adicionales necesitamos?
a-Dos: un contador para recorrer las dos palabras y un contador para las coincidencias
b-Tres: un contador para recorrer cada palabra y un contador para las coincidencias
c-Cuatro: un contador para recorrer cada palabra y un contador para las coincidencias de cada una
Dar la letra correcta, o n si no vale ninguna

29-Se quiere contar cuántas letras de pal1 están también en pal2. ¿Cuántos ciclos hacen falta si usamos la función de búsqueda de caracteres de C, strchr?

30-Si se lee una palabra con scanf("%s",pal); y el usuario teclea "prueba" con las comillas incluidas. ¿Qué queda en pal?

31-Se quiere una condición que sea cierta si pal1 y pal2 son iguales. Darla, usando las variables por ese orden y sin espacios.

32-Para recorrer un texto se utiliza: for(let=0;let<strlen(texto);let++) Si el texto tiene 5 letras, ¿cuántas veces se calcula la longitud del texto?

Respuestas

1 19

2 ab

3 scanf("%c%c%c%c",&l1,&l2,&l3,&l4);

4 tamanho=strlen(dato);

5 printf("%c",palabra[1]);

6 19

7 strstr

8 strncpy

9 texpal[pos-1]

10 sscanf

11 ejemplo

12 palabra[0]=palabra[tam-1];palabra[tam-1]=aux;

13 palabra[tam-1]=letra;

14 pal[0]==LETRA

15 80 81

16 \n

17 0

18 strcmp

19 string

20 b

21 toupper

22 islower

23 if((tex[poslet]=' ')&&(tex[poslet+1]!=' ')

24 ccaa

25 pal[long-let]

26 if((tex[poslet]=let1)&&(tex[poslet+1]==let2)

27 c

28 a

29 1

30 "prueba"

```
31 strcmp(pal1,pal2)==0

32 5
```

Ejemplos resueltos: Textos

Ejemplo de paso a minúsculas

```
1  #include <stdio.h>
2  #include <ctype.h>
3
4  void main()
5  {
6     char cadena[ ] = "ESTO ES UNA CADENA DE PRUEBA";
7     int i;
8
9     for(i = 0; cadena[i]; i++)
10       cadena[i] = tolower(cadena[i]);
11
12    printf("%s\n", cadena);
13 }
```

Descomposición de un código

```
1  #include <stdio.h>
2  /* Para sscanf que lee de una lista de caracteres */
3  #include <string.h>
4  #define TAMAX 21 /* maximo a considerar del codigo:
5  20 caracteres mas terminador */
6
7  void descompon(char codigo[TAMAX], char almacen[TAMAX],
8   int *producto, char lote[TAMAX])
9     {
10    char texprod[TAMAX]; /* Codigo de producto como texto */
11    /* Solo letras excluye las cifras, un numero son solo cifras,
12    el lote es directamente el resto */
13    sscanf(codigo,"%20[^0-9]%20[0-9]%20s",almacen,texprod,lote);
14    /* &* se anulan */
15    sscanf(texprod,"%i",producto);
16    }
17
18 void main()
19    {
20    char codigo[TAMAX],almacen[TAMAX],lote[TAMAX];
21    int producto;
22    printf("Codigo? ");
```

```
23    scanf("%20s", codigo);
24    /* Los resultados que son listas no necesitan & ni * */
25    descompon(codigo,almacen,&producto,lote);
26    printf("Almacen: %s\nProducto: %i\nLote: %s\n",
27    almacen, producto, lote);
28    }
```

El código se lee de una sola vez con %s (ya que no tiene espacios intermedios) y luego se lee sobre él, utilizando `sscanf` con códigos de formato que lee cada uno sólo la parte correspondiente:

1. `%[^0-9]` para leer hasta la primera cifra (excluida)
2. `%[0-9]` para leer sólo cifras
3. `%s` para el resto de caracteres, hasta espacio o fin de línea

Escribir una palabra del revés

```
1  #include <stdio.h>
2  #include <string.h> /* para strlen */
3  #define MAXLETRAS 25
4  void leer(char palabra[MAXLETRAS])
5  /* Lee la palabra
6  ENTRADA:------
7  SALIDA: palabra */
8     {
9     printf("Teclea la palabra: ");
10    scanf("%24s",palabra);
11    }
12 void escribir(char palabra[MAXLETRAS])
13 /* La escribe del reves
14 ENTRADA: palabra
15 SALIDA: --------- */
16    {
17    int cuantas, ultima, letra;
18    cuantas = strlen(palabra);
19    ultima = cuantas - 1;
20    for (letra = ultima; letra >= 0; letra--)
21       {
22       printf("%c",palabra[letra]);
23       }
24    }
25 void main()
26    {
27    char palabra[MAXLETRAS];
28    leer(palabra);
29    escribir(palabra);
30    }
```

Como C no tiene un código para resolver esto directamente, se usa un ciclo que recorre el texto. Es exactamente como un recorrido de lista (pág. 140) porque un texto en C es una lista de caracteres. En este caso el final de la lista es la última letra y para saber su posición necesitamos saber cuántas letras se han cargado en el texto. Eso nos lo dice la función `strlen`

Comprobar paréntesis

Un problema clásico de textos es la comprobación de paréntesis: si están bien equilibrados al abrir y cerrar. Seguramente cualquier programador ha apreciado que en ciertas ocasiones el asunto se puede poner peliagudo.

Una primera fase de razonamiento es que para que esté bien el número de paréntesis que se han abierto tiene que coincidir con el de los que se han cerrado. Pero hay casos en que eso daría como bien y están mal, por ejemplo: `(a+b))-(c-d` Si te fijas, verás que hay dos paréntesis abiertos y dos cerrados, pero está mal, porque el segundo se cierra antes de abrirse. La conclusión es que en ningún momento durante el texto la cuenta de cerrados puede superar a la de abiertos. En el ejemplo anterior, al llegar al segundo paréntesis de cierre, veríamos que sólo hay uno abierto, y ahí tendríamos que marcar error. Se puede hacer también con un sólo contador que suba cuando se abre y baje cuando se cierre. Para dar mensaje más claro de error, iremos sacando el texto letra a letra y nos pararemos al llegar a un posible error.

Cara a la implementación práctica, admitiremos espacios en el texto (formato `%[^\n]`. Tenemos que ir pasando por cada letra del texto, lo que haremos con un ciclo `for` que vaya contando caracteres. Para eso tenemos que saber cuántos tiene el texto, lo que se resuelve con la función `strlen` Cada carácter lo recogemos con `texto[posicion]` y, como es un carácter aislado, lo podemos comparar con el operador de igualdad clásico `==` (por cierto, DOS iguales). Lo compararemos con `'('` o con `')'` (entre apóstrofos porque son caracteres aislados, no textos).

El programa queda:

```
1  #include <stdio.h>
2  #include <locale.h>
3  #include <string.h>  //Para strlen
4
5  void analizar(char texto[100]) {
6  int tam, pos, cuenabre, cuencierra, faltan;
7
8  tam=strlen(texto);
9  for(pos=0, cuenabre=0, cuencierra=0;pos<tam;pos++) {
10         if (texto[pos]=='(') {
11                 cuenabre++;
12         }
13         if (texto[pos]==')') {
14                 cuencierra++;
15         }
16         printf("%c",texto[pos]);// %c porque es un carácter
17         /*La comprobación de que no haya más cerrados que
              abiertos
18         la hacemos en todas las posiciones */
19         if (cuencierra>cuenabre) {
```

```
20                  printf("\nError: se cierra un paréntesis no
                       abierto\n");
21                  return; //No seguimos: ya está mal
22          }
23  }
24  //Falta mirar al final que se hayan cerrado todos
25  if (cuenabre==cuencierra) {
26          printf("\nCorrecto, con %i paréntesis\n",cuenabre);
27  } else {
28          faltan=cuenabre-cuencierra;
29          printf("\nError: %i paréntesis sin cerrar\nabla",faltan
               );
30  }
31  }
32
33  void main()
34  {
35  char texto[200];
36  int i;
37
38  setlocale(LC_ALL,"");
39  printf("Dame el texto: ");
40  scanf(" %[^\n]",texto);
41  analizar(texto);
42  }
```

¿Cómo se trabaja con lotes de datos de tipos variables?

C: Estructuras

En los vectores, todas las componentes son del mismo tipo: todas int, o todas float o todas char. ¿Es posible tener un conjunto de valores de distinto tipo? Por ejemplo, con un componente float y otro char.

La respuesta es sí, pero con condiciones. Los conjuntos de esta clase se llaman **estructuras**. En vez de tener sus componentes numerados, como en los vectores o listas, se le pone un nombre a cada uno; por ejemplo, que el primero se llame `edad` y el segundo se llame `nombre`. Pero la cantidad de componentes tiene que estar determinada de antemano. No es posible tener una etructura con una cantidad de componentes variable de tipos variables.

Ejemplo: datos de un cliente; puede ser una estructura con los componentes nombre (texto), edad (int) y teléfono (int)

Sí se permiten combinaciones de listas y estructuras, que dan bastante juego. En concreto, una lista de estructuras suele sustituir ventajosamente a un conjunto de listas manejadas en paralelo.

Podemos tener una lista de estructuras, y esa lista puede tener tamaño variable, pero cada una de las estructuras componentes está fijada. Por ejemplo, una lista de datos de n clientes, siendo cada dato de cliente lo comentado anteriormente.

También podemos tener una estructura con alguno de sus componentes que sea una lista, que puede ser de tamaño variable. Por ejemplo, unos datos de equipos que incluyan los nombres de sus jugadores, siendo la cantidad de miembros de cada equipo un valor que nos dirán al ejecutar el programa.

Declaración

Vamos a ver dos formas de declarar estructuras: una para cuando todos sus componentes estén totalmente determinados de antemano, y otra para cuando tengamos en la estructura una lista de tamaño variable.

Estructuras totalmente prefijadas

La declaración la haremos en dos pasos: primero especificaremos la forma de la estructura y le daremos un nombre a ese tipo de estructura, y después indicaremos todas las variables que queramos que sean estructuras de ese tipo.

Definición de la estructura. Irá al principio del programa, usando la instrucción `typedef`

Veámoslo con un ejemplo:

```
1 typedef struct {
2         Meses edad;
3         Kg peso;
4 } Barril;
```

En este ejemplo al tipo de estructura que nos interesa se le ha denominado *Barril*, y se le han asignado dos componentes: uno llamado *edad*, que va en `Meses` (habrá sido definido previamente) y otro llamado *peso*, que va en `Kg` (también definidos previamente).

La estructura general de la declaración es:

1. Siempre pondremos una línea `typedef struct {`
2. Una línea por cada componente, con su tipo y su nombre, como si fuera una declaración de una variable independiente. Como `Meses edad;` y `Kg peso;`
3. Terminaremos cerrando la llave y dándole un nombre a ese tipo de estructura (recuerda poner la inicial mayúscula para distinguir que es un tipo, no una variable). Como la línea final del ejemplo: `} Barril;`

Otro ejemplo, incluyendo listas podría ser:

```
1 typedef struct {
2         char nombre[100];
3         char dni[9];
4         int dianac;
5         int mesnac;
6         int anhonac;
7         int telefono;
8 } Persona;
```

Declaración de variables que son estructuras de ese tipo. Es una declaración como otra cualquiera, pero usando el nombre que hemos preparado. Siguiendo con el ejemplo anterior, podría ser:

```
1 Barril blanco, tinto;
```

Ahí hemos declarado que va a haber dos variables del tipo *Barril*: una llamada *blanco* y otra llamada *tinto*.

Como ya hemos dicho podemos usar listas de estructuras. Por ejemplo:

```
1 Barril bodega[100];
```

Ahí hemos declarado que va a haber una lista de 100 estructuras de tipo *Barril*, y esa lista se va a llamar *bodega*. Esto suele ser más cómodo que tener una lista de 100 componentes para las edades y otra para los pesos y tener que llevarlas sincronizadas.

Otros ejemplos, con el segundo tipo de estructuras que habíamos definido:

```
1 Persona proveedor, cliente, plantilla[100];
```

Estructuras que incluyen listas de tamaño ajustado

En este caso no las podemos poner al principio, porque el tamaño aún no se conoce, con lo que la declaración de la variable correspondiente hay que ponerla, junto con la definición de la estructura, donde ya se conozcan los datos.

Aquí se hace pues en un solo paso la definición de la estructura y la declaración de variables. Por ejemplo:

```
1 struct {
2         int codigos[ncomp];
3         Euros precios[ncomp];
4         int cantidad[ncomp];
5         Euros total;
6 } pedido;
```

Ahí hemos definido una variable *pedido* que es una estructura con una lista llamada *codigos* de tamaño *ncomp* (que ya deberá tener cargado su valor en este punto del programa), que son números sin decimales; otra llamada *cantidad*, del mismo tamaño y tipo; otra llamada *precios*, del mismo tamaño, pero del tipo *Euros* (que tenemos que definir al principio del programa); y un componente simple de tipo *Euros* llamado *total*.

Operaciones con estructuras

Las estructuras se pueden copiar en bloque, pero no hacer cálculos en bloque con ellas.

Si queremos copiar una estructura en otra, se hace como en cualquier variable (en esto se diferencian también de las listas, que no se pueden copiar en bloque así).

Ejemplo: `proveedor=cliente;`

Para los cálculos hay que trabajar con cada componente individual. ¿Cómo se indica el componente que nos interesa? Veámoslo sobre un ejemplo:

```
1 blanco.edad=tinto.edad+2;
```

Aquí hemos cogido el componente *edad* de la variable *tinto* (que estaría declarada como estructura del tipo *Barril*), le hemos sumado 2 y el resultado lo hemos metido en el componente *edad* de la variable *blanco*.

En general, para referirnos a un componente de una variable estructura, ponemos primero el nombre de la variable, luego un punto y luego el nombre del componente. Así pues, es completamente distinto al trabajo con listas. Sobre todo, una diferencia trascendental es que una estructura no se recorre con un ciclo, hay que ir operando con cada componente. En la siguiente tabla se subrayan las diferencias entre el trabajo con listas y con estructuras.

	Listas	Estructuras
Indicación de componentes	Corchetes y un valor numérico en medio: lista[1]	Punto y un nombre: estructura.componente;
¿Índices variables?	Sí: `lista[com]`	No
¿Índices calculados?	Sí: `lista[c-1]`	No

¿Recorridos con ciclo?	Sí
	`for(ind=0;ind<n;ind++)`
	`{`
	`lista[ind]=0;`
	`}`
		No
		`estructura.comp1=0;`
		`estructura.comp2=0;`
		`estructura.otro=0;`

¿Copia directa?	No. Ciclo copiando cada componente.
	`for(ind=0;ind<n;ind++)`
	`{`
	`lista1[ind]=lista2[ind];`
	`}`
		Sí `estruc1=estruc2;`

Argumentos de funciones	Siempre de salida	Pueden ser de entrada o salida, como otras variables simples

Paso a funciones

En cuanto a su relación con las funciones, las estructuras se parecen más a las variables normales que a las listas. Sí hay diferencia de que sean de entrada o de salida.

Vamos a comentar los cuatro casos posibles: que se pase la estructura completa o sólo un componente y que sea argumento de entrada o de salida. Veremos un ejemplo y generalizaremos, comentando cómo son la llamada, la definición de la función y el trabajo en ella.

Pasar la estructura completa

Como entrada a la función.

```
1 escribir(fichero,blanco);
```

Se pone sin más el nombre de la variable que sea. Aquí le hemos pasado a la función *escribir* la variable tipo estructura *blanco* (y otra variable diferente).

```
1 void escribir(FILE *fichero,Barril blanco) {
2 ...
3 fprintf(fichero,\" \%i\",blanco.edad);
4 ...
5 }
```

En el ejemplo hemos supuesto que, entre otras cosas, la función escribía en un fichero el componente *edad*.

Entre los paréntesis de la función va, donde le toque, la declaración de la variable, sin ningún cambio. Si la declaración se hubiese hecho sin typedef, habría que repetirla aquí como se hubiese hecho.

Dentro de la función se opera con cada componente con normalidad. Recuerda que como es de entrada, si los cambias, esos cambios se evaporarán al salir de la función.

De salida.

Llamada a la función. Igual que en las variables normales hay que pasar un puntero.

```
1 envejecer(&blanco,nmeses);
```

Aquí le hemos pasado a la función *envejecer* la variable *blanco* para que sea de salida.

```
1 void envejecer(Barril *blanco, Meses nmeses) {
2 ...
3 (*blanco).edad+=nmeses; //Hay que poner los paréntesis
4 //O, sinónimo:
5 blanco->edad+=nmeses;
6 ...
7 }
```

En este caso hemos supuesto que en la función se modificaba un componente de la estructura por lo menos y que esa modificación interesaba mantenerla al salir de la función.

En los paréntesis de la función, la variable va con un * y luego, cuando se usa, hay que ponerle el * y unos paréntesis, para evitar que opere antes el punto que el *, lo que llevaría a error.

Para evitar esta incomodidad, existe un operador "flecha" que tiene el mismo valor y se escribe concatenando un menos y un mayor: `->`

Pasar un componente

Como argumento de entrada.

```
1 escribir(fichero,blanco.edad);
```

Se pone sin más el nombre de la variable con el punto y con el nombre del componente que sea. Aquí le hemos pasado a la función *escribir*, de la variable tipo estructura *blanco* el componente `edad`.

```
1 void escribir(FILE *fichero,int edad) {
2 ...
3 fprintf(fichero," %i",edad);
4 ...
5 }
```

En el ejemplo hemos supuesto que, entre otras cosas, la función escribía en un fichero el componente que le pasamos.

Entre los paréntesis de la función va, donde le toque, la declaración del componente, aislada, sin la estructura a la que pertenece, porque sólo pasa el componente.

Dentro de la función se opera como si fuese una variable sin ninguna referencia a estructuras. Recuerda que como es de entrada, si los cambias, esos cambios se evaporarán al salir de la función.

De salida.

Llamada a la función. Igual que en las variables normales hay que pasar un puntero. Como también va el punto, se recomienda poner los paréntesis para evitar confusión, aunque no es estrictamente necesario.

```
1 envejecer(&(blanco.edad),nmeses);
```

Aquí le hemos pasado a la función *envejecer* de la variable *blanco*, el componente *edad*, para que sea de salida.

```
1 void envejecer(int *edad, Meses nmeses) {
2 ...
3 *edad+=nmeses;
4 ...
5 }
```

En los paréntesis de la función, la variable va con un *, pero declarada sólo como fuese el componente, sin mención a la estructura; y luego, cuando se usa, hay que ponerle el * como a las variables simples que hemos visto otras veces.

Estructuras del sistema para fechas y horas

Aparte de las estructuras que puedas definir tú en tus programas, hay algunas funciones predefinidas que te dan datos en forma de estructuras. Las que vamos a comentar aquí son las de fecha y hora.

Con `#include <time.h>` nos prepara las funciones, macros y tipos para manipular la hora y la fecha del sistema. Define, entre otros, los tipos *clock_t* y *time_t* como representaciones del tiempo en un único número. `clock_t` son comúnmente fracciones de segundo que varían de la centésima a la millonésima, dependiendo del sistema. `time_t` son segundos desde el comienzo del año 1970.

También define el tipo *struct tm* como representación de fecha y hora. Es una estructura con los siguientes componentes:

int tm_sec; segundos después del minuto (0-59)
int tm_min; minutos después de la hora (0-59)
int tm_hour; horas desde la medianoche (0-23)
int tm_mday; día del mes (1-31)
int tm_mon; meses desde enero (0-11)
int tm_year; años desde 1900
int tm_wday; días desde el domingo (0-6)
int tm_yday; días desde el 1 de enero (0-365)
int tm_isdst; indicador de horario de verano

Si declaramos por ejemplo `struct tm tiempo;` para poner el día de la semana a lunes sería: `tiempo.tm_wday=1;`

Función time. La función `time` determina el tiempo en formato `time_t`. Si el tiempo no está disponible, la función retorna el valor -1.

Ejemplo-plantilla:

tiempotime_t=time(NULL);

Función clock. La función *clock* determina el tiempo usado del procesador desde el inicio del programa. Para determinar el tiempo en segundos, el valor retornado por la función *clock* debería ser dividido por el valor CLOCKS_PER_SEC. Si el tiempo usado del procesador no está disponible o su valor no puede ser representado, la función retorna el valor -1.

Ejemplo:

mide=clock();

Función localtime. La función localtime convierte el tiempo de formato `time_t` a formato `struct tm`
Ejemplo:

tiempostruct =localtime(&tiempotime_t);

Función strftime. La función `strftime` convierte el tiempo recogido con `localtime`, en un texto de la forma que se quiera. Para ello tiene unos caracteres de formato específicos para indicar el año, mes, día, etc.
Plantilla:

strftime(texto,tamtexto,formato,tiempoPtr);

tiempoPtr estará declarado como `struct tm *tiempoPtr`El texto nos lo escribirá en texto, cuyo tamaño que habremos declarado, se lo pasamos en tamtexto. Y formato es donde le decimos cómo queremos que nos lo escriba.
Los códigos más interesantes son:

%Y año

%B nombre del mes

%m número del mes

%d día del mes

%A nombre del día de la semana

%H hora

%M minuto

%S segundo

Hay que tener en cuenta que para los nombres usará lo que tenga del sistema.

Función mktime. La función *mktime* convierte un tiempo de formato `struct tm *tiempoPtr`a formato `time_t` (lo contrario de `localtime`). Se usa cuando se han hecho cambios en una fecha y se quiere volver a tener en el otro formato, para operarla.
Plantilla:

tiempotime=mktime(tiempoPtr);

tiempoPtr estará declarado como `struct tm *tiempoPtr`y *tiempotime* como `time_t`
También sirve para calcular y rellenar los datos que falten de una fecha, siempre que sea posible.

Ejemplo de uso

```
1  #include <stdio.h>
2  #include <time.h>
3  #include <locale.h>
4
5  void main() {
6  time_t seg;
7  struct tm *fechahora;
8  char textofecha[80];
9
10 setlocale(LC_ALL,"");
11 seg=time(NULL);
12 printf("¡Desde el 1-1-1970 han pasado %i segundos!\n",seg);
13 fechahora=localtime(&seg);
14 strftime(textofecha,80,"¿Son las %H:%M:%S del %A %d de %B de %Y
      ?\n",fechahora);
15 printf(textofecha);
16 }
```

Ejemplos

Operaciones con quebrados

Hay que hacer un programa que coja dos quebrados (o mixtos), o uno y el resultado anterior si lo hay y opere una de las cuatro operaciones aritméticas, mostrando el resultado. Se repetirá hasta que el usuario elija terminar.

Cada resultado se mostrará simplificado como número mixto; para la simplificación se puede calcular el máximo común divisor por el método de los restos. Este método consiste en que el máximo común divisor de dos números es el mismo que el del menor y el resto de dividir el mayor por el menor, salvo que ese resto sea 0, entonces es directamente el menor. Aplicándolo de forma repetida se puede sacar el máximo común divisor de cualquier pareja de números.

Resolución. El flujo de ejecución del programa va a ser:

- Pedir primer quebrado o mixto
- Pedir segundo
- Pedir operación
- Hacer la que se haya pedido
- Preguntar si terminamos
- Si quiere terminar, cortar
- Preguntar si se quiere aprovechar el resultado previo
- Si no, pedir primer quebrado
- Pedir segundo quebrado

- Pedir operación
- ...

Vamos a poner un ciclo sin condición directamente con la parte que se repite, y lo cortamos desde dentro.

Las operaciones concretas, incluyendo lecturas y escrituras, las haremos en funciones. Tras cada operación habrá que simplificar, antes de escribir el resultado.

En cuanto a la estructura de datos principal es el número mixto, que tiene 3 componentes números sin decimales: parte entera, numerador y denominador.

Con estas ideas planteamos ya el primer borrador de programa:

```
1  #include <stdio.h>
2  #include <locale.h>
3  typedef struct {
4          int entero;
5          int numer;
6          int denom;
7  } Mixto;
8
9  void main() {
10 Mixto quebrado1, quebrado2, resul;
11 char oper,seguir,previo;
12
13 setlocale(LC_ALL,"");
14 pedir(&quebrado1);
15 for(;;) {
16         pedir(&quebrado2);
17         printf("Indica el signo de la operación: ");
18         scanf(" %c",&oper);
19         switch(oper) {
20         case '+':
21                 sumar(quebrado1,quebrado2,&resul);
22                 break;
23         case '-':
24                 restar(quebrado1,quebrado2,&resul);
25                 break;
26         case '*':
27                 multiplicar(quebrado1,quebrado2,&resul);
28                 break;
29         case '/':
30                 dividir(quebrado1,quebrado2,&resul);
31                 break;
32         default:
33                 printf("No se hace nada");
34         }
35         simplificar(&resul);
36         escribir(resul);
```

```
37          printf("¿Continuar? (s/n): ");
38          scanf(" %c",&seguir);
39          if (seguir=='n') {
40                  break;
41          }
42          printf("¿Aprovechar resultado previo? (s/n): ");
43          scanf(" %c",&previo);
44          if (previo=='s') {
45                  copiar(resul,&quebrado1);
46          } else {
47                  pedir(&quebrado1);
48          }
49          //Y vuelta
50  }
51  }
```

Vamo a añadir el resto de funciones.

La de pedir es muy simple, podemos darle un poco de interés con un formato especial para escribir el número: a b/c

```
1  void pedir(Mixto *quebrado) {
2          printf("Dar el mixto como a b/c: ");
3          scanf("%i %i/%i",&(quebrado->entero),&(quebrado->numer),
4          &(quebrado->denom));
5  }
```

Las de las operaciones, las hacemos simples, a lo bruto, porque el resultado ya se simplificará después. Hay distintas formas de enfocarlas; éstas son sólo posibles ejemplos:

```
1  void sumar(Mixto quebrado1,Mixto quebrado2,Mixto *resul) {
2          resul->entero=quebrado1.entero+quebrado2.entero;
3          resul->numer=quebrado1.numer*quebrado2.denom+
4                  quebrado2.numer*quebrado1.denom;
5          resul->denom=quebrado1.denom*quebrado2.denom;
6  }
7
8  void restar(Mixto quebrado1,Mixto quebrado2,Mixto *resul) {
9          resul->entero=quebrado1.entero-quebrado2.entero;
10         resul->numer=quebrado1.numer*quebrado2.denom-
11                 quebrado2.numer*quebrado1.denom;
12         resul->denom=quebrado1.denom*quebrado2.denom;
13 }
14
15 void multiplicar(Mixto quebrado1,Mixto quebrado2,Mixto *resul) {
16 /*
17 (a1+a2/a3)*(b1+b2/b3)=
18 a1*b1+a1*b2/b3+b1*a2/a3+a2*b2/(a3*b3)=
19 a1*b1+(a1*b2*a3+b1*a2*b3+a2*b2)/(a3*b3)
20 */
21         resul->entero=quebrado1.entero*quebrado2.entero;
```

```
22          resul->numer=
23                  quebrado1.entero*quebrado2.numer*quebrado1.denom
                       +
24                  quebrado2.entero*quebrado1.numer*quebrado2.denom
                       +
25                  quebrado2.numer*quebrado1.numer;
26          resul->denom=quebrado1.denom*quebrado2.denom;
27  }
28
29  void dividir(Mixto quebrado1,Mixto quebrado2,Mixto *resul) {
30  /*
31  (a1+a2/a3)/(b1+b2/b3)=
32  ((a1*a3+a2)/a3)/((b1*b3+b2)/b3)=
33  ((a1*a3+a2)*b3)/((b1*b3+b2)/a3)
34  */
35          resul->entero=0;
36          resul->numer=(quebrado1.entero*quebrado1.denom+
37                  quebrado1.numer)*quebrado2.denom;
38          resul->denom=(quebrado2.entero*quebrado2.denom+
39                          quebrado2.numer)*quebrado1.denom;
40  }
```

Para la simplificación, primero reduciremos la fracción a simple (denominador mayor que numerador) y luego buscaremos el máximo común divisor en una función separada.

También hay que tener en cuenta que si al operar, el numerador ha dado 0, entonces no hay simplificación que hacer y podemos dejar el denominador en lo que queramos (1 por ejemplo).

```
1   void simplificar(Mixto *quebrado) {
2   int divisor;
3
4           quebrado->entero+=quebrado->numer/quebrado->denom;
5           quebrado->numer %=quebrado->denom;
6           if (quebrado->numer==0) {
7                   divisor=quebrado->denom;
8           } else {
9                   mcd(quebrado->denom,quebrado->numer,&divisor);
10          }
11          quebrado->numer/=divisor;
12          quebrado->denom/=divisor;
13  }
```

Para el máximo común divisor seguimos las pistas que nos ha dado el enunciado. Se termina cuando se llega a resto 0.

```
1   void mcd(int mayor,int menor,int *maxdiv) {
2   int resto;
3
4           for(resto=mayor%menor;resto!=0;resto=mayor%menor) {
5                   mayor=menor;
6                   menor=resto;
```

```
7           }
8           *maxdiv=menor;
9   }
```

La de escribir es bastante simple, salvo que si el numerador es 0 no escribimos la parte fracción:

```
1   void escribir(Mixto quebrado) {
2           printf("Resultado: %i",quebrado.entero);
3           if (quebrado.numer!=0) {
4                   printf(" %i/%i",quebrado.numer,quebrado.denom);
5           }
6           printf("\n");
7   }
```

Y también es simple la última que nos queda de copiar uno en otro.

```
1   void copiar(Mixto origen,Mixto *destino) {
2           destino->entero=origen.entero;
3           destino->numer=origen.numer;
4           destino->denom=origen.denom;
5   }
```

Agenda

Nos piden un programa que permita cargar datos de personas o buscar personas en los datos cargados. Los datos que tendrá de cada persona son: nombre, apellidos, dirección y teléfono. Los iniciales los leerá del fichero "agenda.txt" y al terminar lo escribirá allí. Las búsquedas serán por cualquier fragmento de los datos.

Resolución. Parece que al principio tiene que leer lo que haya en el fichero, luego cargar datos o hacer búsquedas y luego escribirlo en el fichero. Por lo tanto un primer esquema de bloques del programa puede ser:

- Leer lo que haya en el fichero
- Presentar menú de cargar, buscar o terminar. Hacer lo que se pida y repetir hasta terminar
- Escribir en el fichero

Cada bloque de esos irá en una función. En la de lectura sacaremos la agenda inicial y cuántos son. Para el ciclo de operaciones usaremos un `do-while`, ya que no hay condición que mirar hasta que el usuario no conteste; podemos pedir la elección, por ejemplo, con un número. En la carga cambiará la cantidad, así que será argumento de salida.

Por otro lado vamos a manejar esos datos de gente, así que podemos definir una estructura con ellos. Si queremos buscar por parte del teléfono nos conviene tratarlo como texto, máxime cuando no vamos a hacer operaciones numéricas con ello; aunque son 9 caracteres, necesitamos hueco para el terminador, así que le pondremos tamaño 10. La agenda será una lista de datos de esos. Como puede extenderse, le daremos un tamaño por exceso suficientemente grande.

Este programa podría hacerse sólo con listas: habría una lista de nombres, una lista de direcciones, otra de teléfonos, ... Y tendríamos que manejarlas en paralelo. Puedes hacerlo así si lo prefieres: es también correcto. Pero probablemente una lista de estructuras te resulte más cómodo.

Como manejaremos textos vamos a añadir el `#include <string.h>`

Con estas ideas tenemos nuestro primer borrador de programa:

```
1  #include <stdio.h>
2  #include <locale.h>
3  #include <string.h>
4  typedef struct {
5          char nombre[40];
6          char apellidos[60];
7          char direccion[300];
8          char telefono[10];
9  } Persona;
10
11 void main() {
12 Persona agenda[300];
13 int cuantos,opcion;
14
15 setlocale(LC_ALL,"");
16 leer(agenda,&cuantos);
17 do {
18 printf("1: Cargar datos\n2: Buscar\n3: Terminar\nElige: ");
19 scanf("%i",&opcion);
20 if (opcion==1) {
21         cargar(agenda,&cuantos);
22 }
23 if (opcion==2) {
24         buscar(agenda,cuantos);
25 }
26 }while(opcion!=3);
27 escribir(agenda,cuantos);
28 }
```

Vamos a completar el programa añadiendo esas funciones.

Empezamos por orden de aparición, con la de lectura. Se trata de abrir el fichero y leer repetidamente los datos, contando los que leemos. Si hay un nombre, entonces es que hay apellidos y resto de datos. El nombre vamos a terminarlo con un salto de línea (puede tener espacios); lo mismo los apellidos y la dirección. El teléfono no tendrá espacios.

Cerraremos el fichero al final, porque cuando termine el programa lo reescribiremos.

Entonces podría ser:

```
1  void leer(Persona agenda[300], int *cuantos) {
2  FILE *fichero;
3
4          fichero=fopen("agenda.txt","r");
5          for(*cuantos=0; fscanf(fichero," %[^\n]",agenda[*cuantos
              ].nombre)==1; (*cuantos)++) {
6                  fscanf(fichero," %[^\n]",agenda[*cuantos].
                      apellidos);
```

```
7                   fscanf(fichero," %[^\n]",agenda[*cuantos].
                       direccion);
8                   fscanf(fichero,"%s",agenda[*cuantos].telefono);
9           }
10          fclose(fichero);
11  }
```

En la de cargar empezaremos preguntando cuántos se quieren meter y luego haremos un ciclo añadiéndolos al final de la agenda, parecido al anterior, pero leyendo de pantalla:

```
1   void cargar(Persona agenda[300], int *cuantos) {
2   int anhadir, otro;
3
4           printf("¿Cuántos quieres cargar?");
5           scanf("%i",&anhadir);
6           for(otro=1; otro<=anhadir; otro++, (*cuantos)++) {
7                   printf("Nombre: ");
8                   scanf(" %[^\n]",agenda[*cuantos].nombre);
9                   printf("Apellidos: ");
10                  scanf(" %[^\n]",agenda[*cuantos].apellidos);
11                  printf("Dirección: ");
12                  scanf(" %[^\n]",agenda[*cuantos].direccion);
13                  printf("Teléfono: ");
14                  scanf("%s",agenda[*cuantos].telefono);
15          }
16  }
```

Para buscar se trata de pedir el fragmento que nos den (pensemos que pueda llevar espacios) y averiguar si hay que mirar por nombre o por apellidos o por qué. Para variar, usaremos la inicial (n,a,d o t)Luego en un ciclo recorrer la agenda mirando. Para la comprobación en cada uno, usaremos una función auxiliar; con lo cual, de momento quedará:

```
1   void buscar(Persona agenda[300], int cuantos) {
2   char campo, fragmento[80];
3   int mira;
4
5   printf("Da la inicial del campo que vas a buscar (n,a,d,t): ");
6   scanf(" %c",&campo);
7   printf("Dame el fragmento que voy a rastrear: ");
8   scanf(" %[^\n]",fragmento);
9   for(mira=0;mira<cuantos;mira++) {
10          comprobar(agenda[mira],campo,fragmento);
11  }
12  }
```

En la función de comprobar pasamos el campo a mirar a un texto fijo, por más comodidad y le recorremos buscando la primera letra del fragmento (ciclo hasta encontrar primera letra). Si la localizamos entramos en otro ciclo para ir comprobando las siguientes; este ciclo lo seguimos si se cumplen dos condiciones: aún quedan letras en el fragmento y aún quedan letras en el campo. Además, si comparamos y no coinciden cortamos el ciclo. A su salida comprobamos si hemos

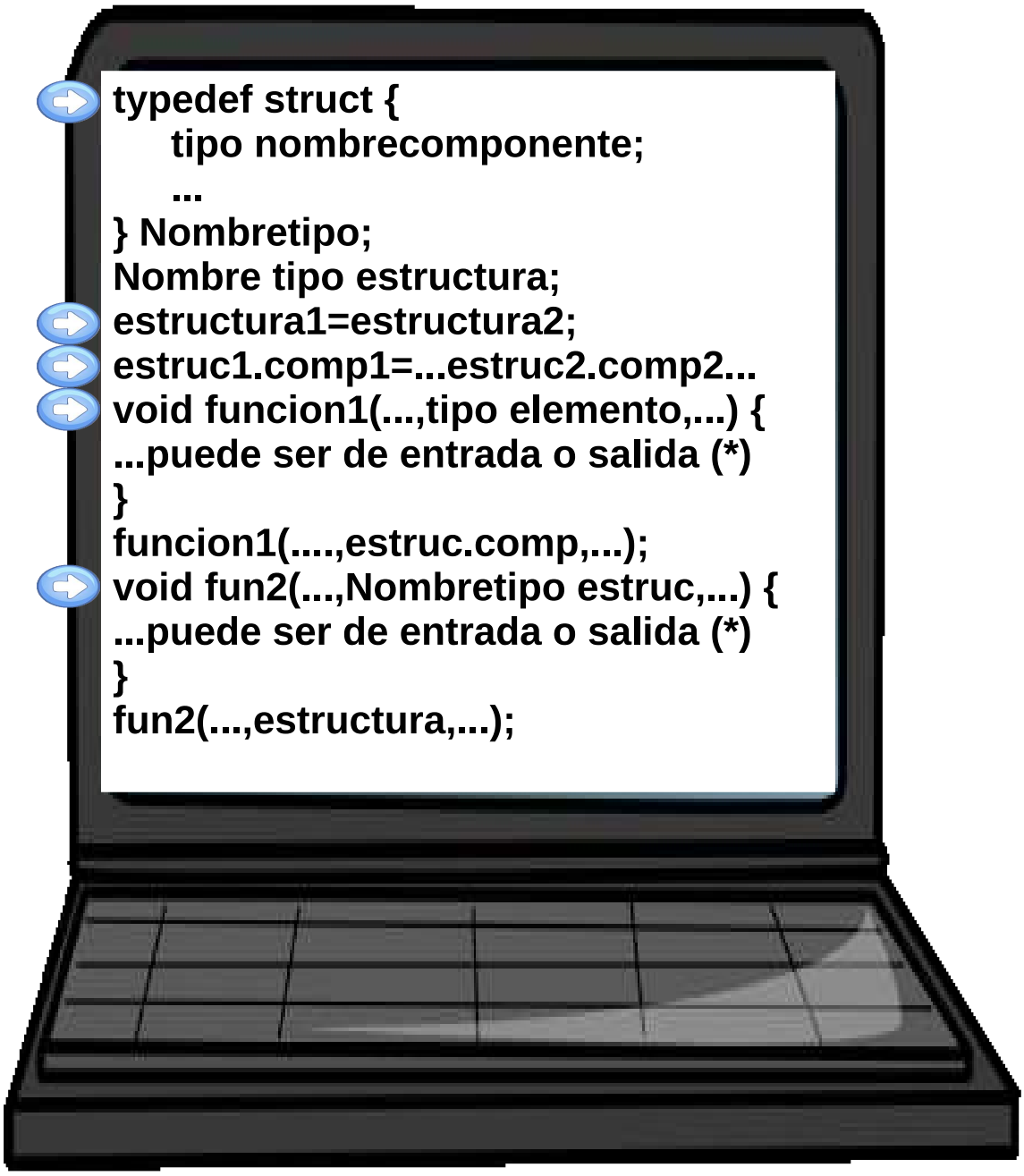
encontrado todas las del fragmento. En caso afirmativo, sacamos todos los datos y salimos de la función.

```
1  void comprobar(Persona datos, char campo, char fragmento[80]) {
2  char amirar[300];
3  int cuantosmirar, cuantofrag, pos1, encampo, enfrag;
4
5  switch(campo) {
6  case 'n':
7          strcpy(amirar,datos.nombre);
8          break;
9  case 'a':
10         strcpy(amirar,datos.apellidos);
11         break;
12 case 'd':
13         strcpy(amirar,datos.direccion);
14         break;
15 case 't':
16         strcpy(amirar,datos.telefono);
17         break;
18 default:
19 //Si no coincide con ningún campo, cortamos
20         return;
21 }
22 cuantosmirar=strlen(amirar);
23 cuantofrag=strlen(fragmento);
24 for(pos1=0;pos1<cuantosmirar;pos1++) {
25         if(amirar[pos1]==fragmento[0]) {
26                 for(encampo=pos1+1,enfrag=1;encampo<cuantosmirar
                       && enfrag<cuantofrag; encampo++,enfrag++) {
27                         if(amirar[encampo]!=fragmento[enfrag]) {
28                                 break;
29                         }
30                 }
31                 if(enfrag==cuantofrag) {
32                         printf("Nombre: %s %s\nDireccion: %s\
                              nTelefono: %s\n",
33                         datos.nombre,datos.apellidos,datos.
                              direccion,
34                         datos.telefono);
35                         return;
36                         }
37         }
38 }
39 }
```

Finalmente, la función de escribir es más o menos la simétrica de la de leer:

```
1  void escribir(Persona agenda[300], int cuantos) {
```

```
2  FILE *fichero;
3  int toca;
4
5  fichero=fopen("agenda.txt","w");
6  for(toca=0;toca<cuantos;toca++) {
7          fprintf(fichero,"%s\n%s\n%s\n%s\n",
8          agenda[toca].nombre,
9          agenda[toca].apellidos,
10         agenda[toca].direccion,
11         agenda[toca].telefono);
12 }
13 }
```


typedef struct {
tipo nombrecomponente;
...
} Nombretipo;
Nombre tipo estructura;
estructura1=estructura2;
estruc1.comp1=...estruc2.comp2...
void funcion1(...,tipo elemento,...) {
...puede ser de entrada o salida (*)
}
funcion1(....,estruc.comp,...);
void fun2(...,Nombretipo estruc,...) {
...puede ser de entrada o salida (*)
}
fun2(...,estructura,...);

¿Cómo se trabaja con matrices o tablas?

Declarar, operar, pasar a funciones

C: Tablas o arrays multidimensionales

Recursos en el aula virtual de tablas

Teoría

Aparte de estos apuntes

Ejercicios iniciales de instrucciones concretas

Para comprobar que has interiorizado la teoría
Y recuerda las cuestiones de autoevaluación que tienes aquí en el apartado III

Ejemplos explicados

Con la solución y explicaciones
Y también en estos apuntes, en el apartado 0.14

Ejemplos activos

Visualizaciones de la ejecución paso a paso de un programa

Ejemplos con solución

La solución sin o con pocas explicaciones
Y también en estos apuntes, en el apartado III

Ejercicios con pistas

Sin memorización de por dónde vas

Ejercicios propuestos

Sin la solución

Controles y exámenes de otros años

Algunos con solución, como ejemplos resueltos, y otros sin ella, como ejercicios propuestos

Las tablas son conjuntos de valores ordenados según dos dimensiones: filas y columnas. Matemáticamente se llaman matrices. Pueden considerarse listas de listas. Serán valores todos del mismo tipo y que se van a tratar conjuntamente.

En el caso de caracteres, una tabla equivale a una lista de palabras o de frases.

Una tabla o array multidimensional se comporta de manera similar a uno unidimensional, por lo que se aplica todo lo comentado sobre inicialización, asignación, etc. La diferencia principal está en el paso de parámetros a una función en el que es preciso especificar todas las dimensiones pudiendo

omitirse la primera opcionalmente. Como puede surgir la confusión, se recomienda darlas todas siempre.

Una tabla se declara según:

```
tipo_dato vector[num_filas] [num_columnas];
```

Ejemplo:

```
1  //Funcion que calcula el producto de dos matrices cuadradas.
2
3  void multiplicar(float a[DIMENSION][DIMENSION], float b[
       DIMENSION][DIMENSION],
4   float c[DIMENSION][DIMENSION]) {
5  int i, j, k;
6  for(i = 0; i < DIMENSION; i++)
7    for(j = 0; j < DIMENSION; j++)
8    {
9      c[i][j] = 0.0;
10         for(k = 0; k < DIMENSION; k++) {
11                 c[i][j] += a[i][k] * b[k][j];
12         }
13   }
14 }
```

Plantillas: Tablas

Declaración

Plantilla:

```
tipo nombre[nºfilas][nºcolumnas]
```

Ejemplo:

```
1  int clients[horas][dias];
```

Se recuerda que cuando el número de filas o de columnas es una variable la declaración se pone cuando el valor de esa variable ya es conocido

Uso

Elemento individual

Plantilla:

```
..nombretabla[fila][columna]..
```

Ejemplo:

```
1  if(clients[horya][hoy]==0)
```

Fila

Plantilla:

```
...nombretabla[fila]...
```

Ejemplo:

```
escribe(dias,clients[horya]);
```

Tabla completa

Plantilla:

```
...nombretabla...
```

Ejemplo:

```
lee(horas,dias,clients);
```

Lectura

Para el caso de lecturas típicas se ha preparado una instrucción (técnicamente es una *macro*), que lee consecutivamente por filas el conjunto de valores.

Plantilla para el caso de tabla de números:

```
letanu(formato,tabla,cantidaddefilas,cantidadcolumnas);
```

Ejemplo:

```
letanu("%f",ventas,puestos,dias);
```

Plantilla para el caso de lista de palabras:

```
lelispa(listadepalabras,cantidaddepalabras);
```

Ejemplo:

```
lelispa(mispalabras,cuantas);
```

Escritura

Para el caso de escrituras típicas se ha preparado una instrucción (técnicamente es una *macro*), que escribe por filas el conjunto de valores.

Plantilla para el caso de tablas de números: Conviene tener en cuenta en el formato la separación entre valores dentro de cada fila. El salto al final de cada fila ya está incluido.

```
estanu(formato,tabla,cantidaddefilas,cantidadcolumnas);
```

Ejemplo:

```
estanu("%f ",ventas,puestos,dias);
```

Plantilla para el caso de listas de palabras:

```
eslispa(listapalabras,cantidaddepalabras);
```

Ejemplo:

```
eslispa(nombres,numamigos);
```

Recorridos por tabla

El proceso a hacer con cada elemento de la tabla puede ser cualquiera. En el ejemplo se presenta una lectura de tabla desarrollada (hace lo mismo que la macro antedicha). Si se usara `printf` en lugar de `scanf`, se tendría una escritura de tabla.

Plantilla:

```
for(fila = 0; fila < nº de filas; fila++) {
          for(columna = 0; columna < nº de columnas; columna++) {
                  instrucciones procesando tabla[fila][columna]
          }
   }
```

Ejemplo:

```
1 printf("Dar tabla: ");
2 for(fila = 0; fila < totfilas; fila++) {
3         for(colum = 0; colum < totcols; colum++) {
4                 scanf("%f",&tabla[fila][columna]);
5         }
6 }
```

Ordenación

Como se vio con las listas, el lenguaje C trae implementada una función para ordenar (`qsort`). Esta función requiere que los elementos a ordenar estén contiguos (por ejemplo filas de una tabla). Si no es el caso (por ejemplo columnas de una tabla) habría que programar toda la ordenación. Vamos a ver el caso de filas de una tabla ordenadas según el valor de cierta columna (`posicion`), que tendría que ser en este caso, excepcionalmente, una variable global, es decir, declarada al principio del programa, con las constantes.

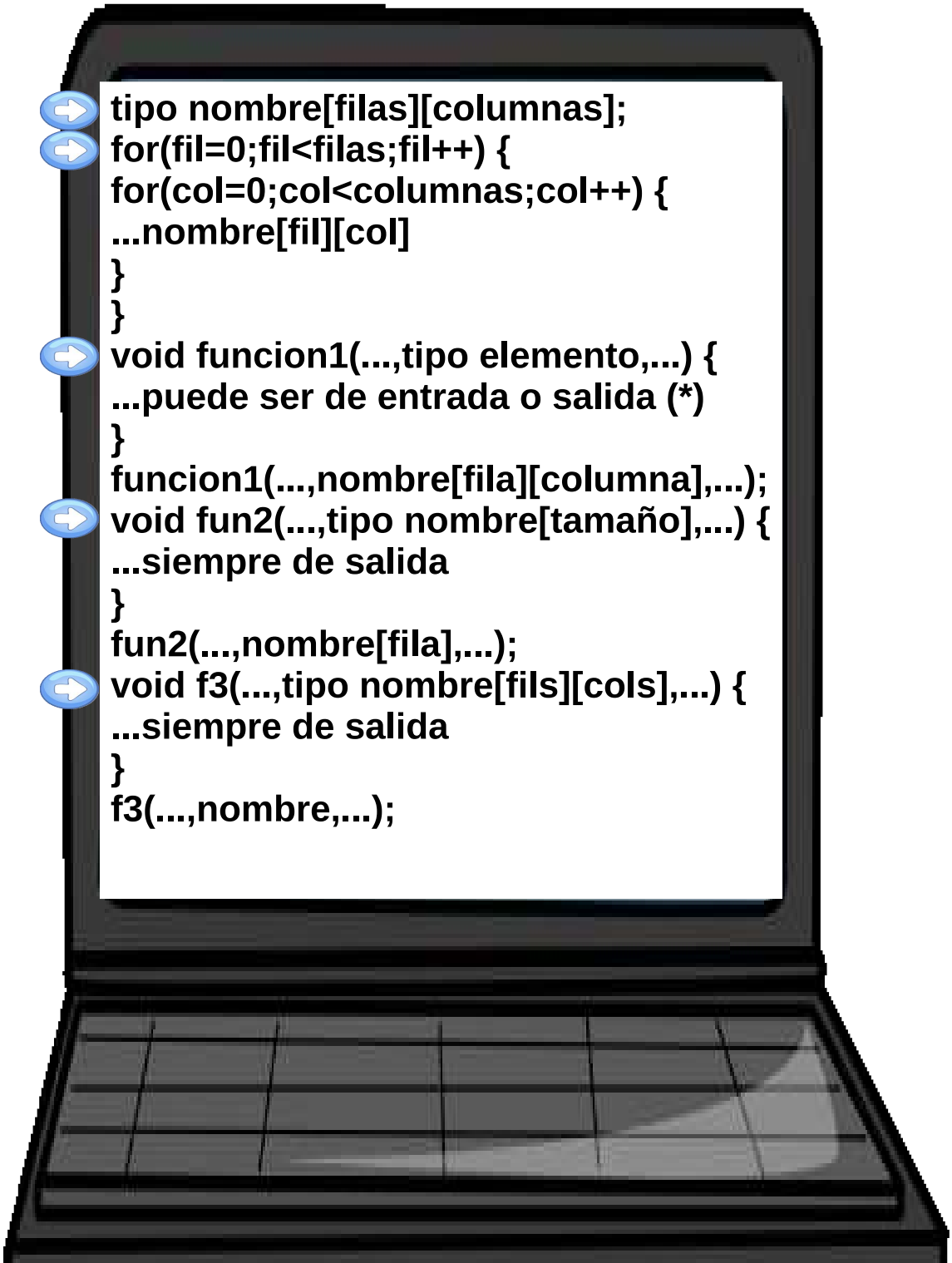
Plantilla:

```
tipo nombrefun(const void *f1, const void *f2) {
   const tipo *fila1, *fila2;
   fila1=(tipo *)f1;
   fila2=(tipo *)f2;
   return fila1[posicion]-fila2[posicion];
   }
   ...
   qsort(tabla,numerofilas,numerocolumnas*sizeof(tipo),&nombrefun);
   ...
```

Ejemplo:

```
1 int intcomta(const void *f1, const void *f2) {
2 const int *fila1, *fila2;
3 fila1=(int *)f1;
4 fila2=(int *)f2;
5 return fila1[0]-fila2[0];
```

```
6 }
7 ...
8 qsort(tabla,numerofilas,numerocolumnas*sizeof(int),&intcomta);
9 ...
```


tipo nombre[filas][columnas];
for(fil=0;fil<filas;fil++) {
for(col=0;col<columnas;col++) {
...nombre[fil][col]
}
}
void funcion1(...,tipo elemento,...) {
...puede ser de entrada o salida (*)
}
funcion1(...,nombre[fila][columna],...);
void fun2(...,tipo nombre[tamaño],...) {
...siempre de salida
}
fun2(...,nombre[fila],...);
void f3(...,tipo nombre[fils][cols],...) {
...siempre de salida
}
f3(...,nombre,...);

Ejemplos de instrucciones

Aquí se presentan ejemplos de instrucciones de lo que hemos visto.

```
Metros3 co2[nvehic][nper];
   base[if][ic]=0;
   printf("A distancia %i se llega en
      %f\n",cuadro[selc][selc2],medt[selc][selc2]);
   Calorias gasto[DEP][ED];
   for(c=0;c<npla;c++){    printf("%i\n",ener[0][c]);}
   altura = forja[0][nmax-1] + forja[1][nmax-1];
   anular(&metrica[0][0]);
   void potenmat(int veces, int nf, int nc, float dato[nf][nc]) {...}
   for(c=0;c<ndias;c++){   scanf("%f",&riego[npsel][c]);}
   printf("%f ",densid[abs][or]);
   char mensaje[n][TERMC];
   scanf("%i %f",&codigo[isup][iprod],&precio[isup][iprod]);
   veloc[t][tiem] -= acelf;
   Pendiente escal[ncent][nacc];
   scanf("%i",&clase[cc][cr]);
   mostrar(catal[np][nl]);
```

Cuestiones de autoevaluación

Preguntas

1-Un programa tiene que calcular el factorial de cada uno de los números de una matriz. Contiene una función que calcula el factorial de un número, cuya definición empieza así:
void factorial(int numero, int *resul)
Dar la instrucción C (sin espacios) que, usando esta función, mete en la variable aux el factorial del elemento fila i columna j de la tabla llamada matriz.

2-Dar la llamada a la función cuya linea inicial de definición es:
void traspon(int filas,int cols, int matriz[filas][cols])
Admitir que en la función que llama las variables se llaman igual que en esta función.Poner sólo los espacios absolutamente imprescindibles.

3-Dar la definición de una lista de nombres llamada lisnombres que serán como máximo CANTIMAX y cada uno puede tener hasta LETRASMAX letras (incluyendo el hueco de terminación). Poner sólo los espacios imprescindibles. No se declaran más variables en esa instrucción.

4-Un programa lee una lista de
nombres de personas, edades y cursos en que están matriculados. Tiene que decir cuántas personas están en un curso que pida el usuario y

calcular la media de edades. ¿Cuántas listas y cuántas tablas necesita? Dar los dos números por ese orden separados por un espacio.

5-Un programa gestiona un diccionario inglés/español, dando la traducción a la palabra que le dé el usuario. Todas las variables que usa son de tipo char ¿Cuántas son listas y cuántas son tablas?. Suponer que el resultado se mete en una variable. Dar los dos números por el orden pedido separados por un espacio.

6-En cierto programa
hay que poner a cero los valores de la diagonal de una matriz de números sin decimales que se llama matnum. El ciclo se define comofor(fila=0;fila<totfilas;fila++)Dar la única instrucción qeu va dentro de este ciclo, sin espacios.

7-Un programa tiene que leer dos matrices cuadradas de la misma dimensión y escribir su suma y producto. ¿Cuál es el número mínimo de tablas o matrices que hay que declarar?

8-Para recorrer todos los elementos de una matriz hacen falta ¿cuántos ciclos hacen falta, uno dentro de otro?

9-Para almacenar una lista de palabras se ha declarado la siguiente variables:
char lispal[cuanpal][MAXLETRAS];
Dar la instrucción (sin espacios) que mete en la variable tope el tamaño de la primera palabra de la lista.

10-Un programa tiene que gestionar la ocupación las butacas de un local. El usuario dirá el número de filas y cuántas butacas hay en cada fila. ¿Qué variable se usará, lista o tabla?

11-Un programa tiene que leer una serie de datos de ventas que incluyen: número de vendedor, nombre de vendedor, mes del año, importe en euros. ¿Cuántas listas y cuántas tablas necesita para esto? Dar los números por ese orden separados por un espacio.

12-Si un programa tiene que recorrer una matriz usa un ciclo dentro de otro. Si el externo es el de filas, podría ser:

```
for (fil = 0; fil < filmax; fil++)
```

En cierto programa hay que recorrer todas las filas excepto la filno. Se tienen dos opciones:

1- Poner la condición en el ciclo: `for (fil=0; (fil < filmax) && (fil != filno); fil++)`

2- Dejar el ciclo como está y meter lo que hay dentro dentro de un if cuya cabecera sea: `if (fil != filno)`

Decir cuál de estas opciones es válida con su número (ó 12 si ambas valen

ó 0 si no vale ninguna)

13-Dar una instrucción que pone el valor que está en la variable mat, fila fil y columna col en su simétrico respecto a la diagonal. No poner espacios.

14-Se quiere dar un valor x a todos los elementos de un trozo de matriz y otro valor z al resto. Se plantean las siguientes estrategias alternativas:
a-Recorrer el trozo dando valor x y después recorrer toda la matriz dando el valor z
b-Recorrer toda la matriz dando el valor z y luego recorrer el trozo dando el valor x
c-Recorrer toda la matriz dando el valor z con condicionales para saltarse el trozo y luego recorrer el trozo dando el valor x
d-Recorrer el trozo dando valor x y después recorrer toda la matriz dando el valor z con condicionales para saltarse el trozo
Dar las letras de las válidas por el orden presentado, o n si ninguna vale

15-Se tiene la matriz mat, con MAXFIL filas y MAXCOL columnas. Escribir la condición que iría dentro de los paréntesis de un if que quisera comprobar si el último elemento de la fila fil es igual que el primero de la siguiente. No poner espacios.

16-Dar la condición que iría entre los paréntesis de un if para comprobar si el elemento de la primera fila, primera columna de la tabla tabnum es negativo. No poner espacios.

17-Se quiere escribir la suma de los elementos de cada fila de una matriz. Para ello se pone un ciclo que recorre las filas de la matriz y dentro otro que recorre las columnas. ¿En qué sitio hay que poner el acumulador a 0? y ¿en qué sitio hay que poner la instrucción de escritura? Las posibilidades son:
a-Antes de los dos ciclos
b-Dentro del de filas, pero antes del de columnas
c-Dentro del de columnas
d-Dentro del de filas, después del de columnas
c-Después de los dos ciclos
Dar las dos letras para las dos preguntas juntas, por el orden preguntado.

18-Se quiere almacenar información de dónde hay objetos en una cuadrícula para un juego. Estrategias alternativas:
a-Lista de filas y lista de columnas de la posición de cada objeto
b-Matriz con valor de objeto en su posición correspondiente o 0 si no hay objeto
c-Tabla que tiene en su primera columna las filas y en su segunda las columnas de los objetos

Dar la(s) válida(s), seguidas y por el orden presentado si son varias, o n si ninguna vale

19-Se lee una lista de palabras en una tabla llamada lispal. El usuario mete las siguientes: "elefante", "tigre", "toro" y "oso". Luego el programa tiene un ciclo que recorre las cuatro filas de lispal con la variable fila. Dentro de ese ciclo hay sólo una instrucción:
printf("%c",lispal[fil][0]);
¿Qué saldrá en pantalla?

20-Se lee una lista de palabras en una tabla llamada lispal. El usuario mete las siguientes: "elefante", "tigre", "toro" y "okapi". Luego el programa tiene un ciclo que recorre las cuatro filas de lispal con la variable fila. Dentro de ese ciclo hay sólo una instrucción:
printf("%c",lispal[fil][fil]);
¿Qué saldrá en pantalla?

21-Se quiere almacenar una serie de vectores. La cantidad de vectores es a elegir por el usuario. El tamaño de cada vector también y son distintos. Estrategias alternativas:
a-Varias listas de tamaño fijo suficientemente grande
b-Varias listas, cada una de tamaño adaptado a lo que pida el usuario
c-Una tabla de tamaños fijos suficientemente grande
d-Una tabla de filas adaptada a lo que pida el usuario y columnas fijas suficientemente amplia
Dar la opción válida, o varias seguidas y por el orden propuesto si es el caso, o n si ninguna vale

22-Se quiere trabajar con una serie de listas. La cantidad de listas es a elegir por el usuario. El tamaño de cada lista también y son distintos. Aparte de las propias listas, para el trabajo posterior hay que almacenar:
a-Basta con el mayor de los tamaños
b-El tamaño de cada una
c-La cantidad de listas y el mayor de los tamaños
d-La cantidad de listas y el tamaño de cada una
Dar la opción válida, o varias seguidas y por el orden propuesto si es el caso, o n si ninguna vale

23-Un programa tiene que generar una tabla a partir de números que le dé el usuario. El número de filas va a ser tres y el máximo de columnas lo dirá el usuario. Cada número se añadirá en una fila u otra según su valor (hay un intervalo posible para cada fila). ¿Qué variables hacen falta además de los intervalos de cada fila?
a-Tres listas del tamaño máximo, una para cada fila
b-Tres listas y tres contadores para saber cuántos números han entrado en cada una

c-Una tabla de tres filas y columnas según el máximo
d-Una tabla de tres filas y columnas según el máximo, y tres contadores para saber cuántos números han entrado en cada fila
Dar las letras de las opciones válidas juntas, o n si ninguna vale

24-Un programa tiene que comprobar todos los elementos que están por debajo de la diagonal (excluida). Para ello usa un ciclo que barre las filas con la variable fil y otro que recorre las columnas necesarias:
for(col=0;____;col++)
¿Qué va en el hueco? Ponerlo sin espacios

25-En un programa se tiene una tabla mat, de tamaño filas y cols y se quiere acumular el producto de todos los elementos de la fila fil en el último (que quedará machacado). Para ello se pone el ciclo:
for(col=0;col<cols-1;col++)
¿Qué instrucción va dentro del for? Usar el operador *= y no poner espacios.

26-Se quiere comprobar si cierto elemento de una tabla es el máximo de su fila y también el máximo de su columna. ¿Cuántos ciclos hacen falta?

27-En la tabla de texto lispal se tiene almacenada en cada fila el texto de respuesta que hay que dar si el usuario indica el número de esa fila, que está en filped. Dar la instrucción que escribe esa respuesta, usando printf y sin espacios.

28-Un programa tiene que buscar la fila de una matriz que da máximo cuando se suman los valores absolutos de sus elementos. ¿Cuántos ciclos harán falta?

29-Un programa almacena una serie de valores en una tabla de filas a indicar por el usuario, pero el número de columnas ocupadas varía en cada fila, por lo que se opta por almacenar separadamente el número de columnas utilizadas en cada fila. Para guardar estos valores se plantean las siguientes estrategias alternativas:
a-Una columna adicional en la tabla
b-Una lista
c-Una serie de variables simples individuales
Dar las opciones válidas juntas, o n si no vale ninguna

30-Se leen gastos en una serie de productos a indicar por el usuario en una serie de periodos también a indicar. Se usa una tabla llamada gas poniendo productos en filas y periodos en columnas. El usuario da valores desordenados, dando cada vez el producto (variable prod), el periodo (variable per) y el gasto. El producto y el periodo los numera a partir de 1. Dar la instrucción de lectura del gasto, cargándolo en la tabla, con un campo de formato %f y sin espacios.

31-En un programa hay que sumar cada fila de una tabla, llamada tab que tiene nfil filas y ncol columnas. Para ello se dispone de una función que hace la suma de una lista, cuya cabecera es: void sumar(int cuantos, int lista[cuantos], int *suma)
Poner la llamada a esta función para obtener la suma de la fila f en el componente f de la lista lisum.

32-Se va a manejar una lista de datos personales que son: nombre, número, DNI, domicilio, categoría (hay 5), edad y sección (son 4 letras). Decir cuántas listas y cuántas tablas hacen falta, separando los números con un espacio.

33-Se va a manejar una lista de datos personales que son: nombre, peso, estatura, edad y número de hijos. Se quiere usar una única gran tabla para almacenar los datos. Decir qué condición hay que exigir para que sea posible:
a-Siempre será posible
b-Que el peso sea sin decimales
c-Que el nombre no tenga espacios
Dar la condición o condiciones juntas necesarias por el orden presentado, o n si no vale ninguna

34-Dar la declaración de una tabla llamada mat para números con decimales, con 3 filas y 5 columnas. No poner espacios más que los imprescindibles. En esa instrucción no se declarará ninguna otra variable.

35-Un programa tiene que leer una matriz, sumar todos sus elementos y escribir el resultado. Para la suma usa una función calsuma, cuyos datos son el número de filas, el de columnas, la tabla y da como resultado la suma. Si los nombres de las variables correspondientes son nfil, ncol, tab y sum, dar la llamada a la función, sin espacios.

36-¿Qué función C se utiliza para copiar una tabla entera?

Respuestas

```
1 factorial(matriz[i][j],&aux);
```

```
2 traspon(filas,cols,matriz);
```

```
3 char lisnombres[CANTIMAX][LETRASMAX];
```

4 2 1

5 2 2

6 matnum[fila][fila]=0;

7 3

8 2

9 tope=strlen(lispal[0]);

10 tabla

11 3 1

12 2

13 mat[col][fil]=mat[fil][col];

14 bcd

15 mat[fil][MAXCOL-1]==mat[fil+1][0]

16 tabnum[0][0]<0

17 bd

18 abc

19 etto

20 eirp

21 cd

22 d

23 bd

24 col<fil

25 mat[fil][cols-1]*=mat[fil][col]

26 2

27 printf("%s",lispal[filped]);

28 3

```
29 ab

30 scanf("%f",&gas[prod-1][per-1]);

31 sumar(ncol,tab[f],&lisum[f]);

32 4 3

33 n

34 float mat[3][5];

35 calsuma(nfil,ncol,tab,&sum);

36 memcpy
```

Ejemplos resueltos: Tablas

Calificaciones

Se tiene una lista de estudiantes con unas calificaciones en una serie de exámenes de una asignatura. Cada examen tiene un cierto valor en la nota final y se desean obtener las notas finales.

Resolución

Vamos a plantearnos la resolución del enunciado en 3 versiones:

1. Un sólo alumno, todos los exámenes valen igual
2. Un sólo alumno, notas ponderadas
3. Versión completa: varios alumnos, cada examen con su valor

Un sólo alumno, exámenes de igual valor. En este caso tenemos una lista de notas, de la que hay que sacar la media. Este caso es el mismo que hemos resuelto con una función en la página 155, por lo que vale lo dicho allí.

El programa tendría una lectura de datos, usaría esa función de media y tendría una escritura de resultados.

Un sólo alumno, cada examen un valor. Aquí hay dos listas: la de notas y la de valor de cada examen. La nota resultado es:

$$\frac{\sum n_i v_i}{\sum v_i}$$

donde n_i son las notas de cada examen y v_i son los valores de cada uno. La diferencia con la versión anterior es doble:

- No se suman simplemente las notas, sino las notas multiplicadas por los valores

- No se divide por cuántos son, sino por la suma de valores

La primera diferencia es fácil de implementar, dada la lista de valores. Para la segunda hay que hacer una suma. Podemos usar una plantilla de acumulación a partir de una lista, pero, teniendo hecha la parte de la media, realmente nos puede servir de plantilla, añadiendo la suma de estos datos. Para indicar el término ter de la lista lis, usaremos la notación lis[ter], que es como luego habrá que ponerlo en C. La función de media quedaría:

Poner total a 0
Poner sumavalores a 0
para todos los casos de posición **repetir**
  Sumar lista[posicion]×valores[posicion] a total
  Sumar valores[posicion] a sumavalores
final de **repetir**
Dividir total por sumavalores

Varios alumnos, cada examen con su valor

En este caso cambia la estructura de los datos. Antes teníamos una lista de notas y una lista de valores de exámenes. Ahora tenemos una **tabla** de notas y una lista de valores de exámenes. Vemos que el proceso de cálculo de cada media es usar la función ya desarrollada con cada fila de la tabla, así pues hay que hacer un recorrido por la tabla procesando cada fila. La notación que usaremos para indicar una fila de una tabla es análoga a la que hemos usado para indicar un elemento de una lista. Basándonos en la plantilla de recorrido de tablas (pág. 204), tendríamos una función de cálculo que usaría la de media anterior:

para todos los casos de fila de tablanotas **repetir**
  Calcular media con tablanotas[fila]
final de **repetir**

Paso a C

Vamos a dividir en funciones todo lo posible, usando los bloques que hemos comentado:

- Lectura de datos: función aparte
- Media de cada alumno: función aparte
- Recorrido por los alumnos organizando el cálculo y escribiendo resultados: lo dejaremos en `main`

Lectura de datos. Los cogemos de un fichero cuyo nombre preguntamos al usuario. Primero leemos las cantidades, luego la lista de valores de cada examen y luego la tabla de notas de los alumnos. Esto es un recorrido de tabla leyendo de fichero (pág. 204).

```
1  printf("¿Nombre del fichero?\n");
2  scanf(" %s",nomfich);
3  fichero=fopen(nomfich,"r");
4  fscanf(fichero," %i %i",&cuannotas,&cuangente);
5  for(posic=0;posic<cuannotas;posic++) {
6          fscanf(fichero," %f",&valores[posic]);
7  }
8  for(fila = 0; fila < cuangente; fila++) {
9          for(colum = 0; colum < cuannotas; colum++) {
10                 fscanf(fichero," %f",&tablanotas[fila][colum]);
11         }
12 }
```

Variables.

Variable	Tipo	Argumento
nomfich	Letras. Para el tamaño supondremos que no habrá nombres de fichero que lleguen a las 40 letras.	Sólo se usa dentro de la función: variable local
fichero	Fichero	Sólo se usa dentro de la función: variable local
cuannotas	Sin decimales, es un recuento	Se le da valor en la función y lo usamos después: argumento de salida
cuangente	Sin decimales, es un recuento	Se le da valor en la función y lo usamos después: argumento de salida
posic	Sin decimales, es una posición	Sólo se usa dentro de la función: variable local
valores	Lista de valores posiblemente con decimales. El tamaño le daremos por exceso, 20.	Se le da valor en la función y lo usamos después: argumento de salida (aunque en listas da igual que sea de entrada o de salida)
fila	Sin decimales, es una posición	Sólo se usa dentro de la función: variable local
colum	Sin decimales, es una posición	Sólo se usa dentro de la función: variable local
tablanotas	Tabla de valores posiblemente con decimales. El tamaño le daremos por exceso, 20 para las notas (columnas) y 300 para las filas (gente).	Se le da valor en la función y lo usamos después: argumento de salida (aunque en tablas da igual que sea de entrada o de salida)

```
1  void leedatos(int *cuannotas, int *cuangente, float valores[20],
       float tablanotas[300][20]) {
2  char nomfich[40];
3  FILE *fichero;
4  int posic,fila,colum;
5
6  printf("¿Nombre del fichero?\n");
7  scanf(" %s",nomfich);
8  fichero=fopen(nomfich,"r");
9  fscanf(fichero," %i %i",&*cuannotas,&*cuangente);
10 for(posic=0;posic<*cuannotas;posic++) {
11         fscanf(fichero," %f",&valores[posic]);
12 }
13 for(fila = 0; fila < *cuangente; fila++) {
14         for(colum = 0; colum < *cuannotas; colum++) {
15                 fscanf(fichero," %f",&tablanotas[fila][colum]);
16         }
```

```
17 }
18 }
```

Media de cada alumno. Se corresponde con las ideas de (pág. 214).

```
1 total = 0;
2 sumavalores = 0;
3 for(posicion=0;posicion<cuannotas;posicion++) {
4         total+= lista[posicion]*valores[posicion];
5         sumavalores+= valores[posicion];
6 }
7 media= total / sumavalores;
```

Variables.

Variable	Tipo	Argumento
total	Con decimales, porque es una suma de números con decimales	Sólo se usa dentro de la función: variable local
sumavalores	Con decimales, porque es una suma de números con decimales	Sólo se usa dentro de la función: variable local
posicion	Sin decimales, es una posición	Sólo se usa dentro de la función: variable local
cuannotas	Sin decimales, es un recuento	Llega desde main, no se modifica en la función: argumento de entrada
lista	Lista de valores posiblemente con decimales. El tamaño hay que mantener la decisión tomada anteriormente, 20.	Llega desde main, no se modifica en la función: argumento de entrada (aunque en listas da igual que sea de entrada o de salida)
valores	Lista de valores posiblemente con decimales. El tamaño hay que mantener la decisión tomada anteriormente, 20.	Llega desde main, no se modifica en la función: argumento de entrada (aunque en listas da igual que sea de entrada o de salida)
media	Con decimales	Se le da valor en la función y lo usamos después: argumento de salida

```
1 void mediacada(int cuannotas, float lista[20], float valores
     [20], float *media) {
2 float total,sumavalores;
3 int posicion;
4
5 total = 0;
6 sumavalores = 0;
7 for(posicion=0;posicion<cuannotas;posicion++) {
8         total+= lista[posicion]*valores[posicion];
```

```
9           sumavalores+= valores[posicion];
10  }
11  *media= total / sumavalores;
12  }
```

main. Nos basamos en las decisiones que hemos tomado en las funciones anteriores y en las ideas del recorrido de alumnos (pág. 215). Vamos a escribir los resultados en otro fichero cuyo nombre preguntaremos al usuario, de forma análoga a lo que hicimos para leer los datos.

```
1   void main() {
2   char nomfich[40];
3   FILE *fichero;
4   int cuannotas,cuangente,fila;
5   float valores[20], tablanotas[300][20], media;
6
7   setlocale(LC_CTYPE,"");
8   printf("¿Nombre del fichero para escribir las medias?\n");
9   scanf(" %s",nomfich);
10  fichero=fopen(nomfich,"w");
11  leedatos(&cuannotas, &cuangente, valores, tablanotas);
12  for(fila=0;fila<cuangente;fila++) {
13            mediacada(cuannotas, tablanotas[fila], valores, &media)
                ;
14            printf(" %f\n",media);
15  }
16  }
```

Interpolación

Dada una serie de valores de una variable y los de una cierta función en ellos, obtener mediante interpolación lineal el valor de la función en otro valor dado por el usuario. Los valores donde la función es conocida son dados sin ningún orden.

La fórmula de interpolación lineal es:

$$\frac{x_{sup}y_{inf} - x_{inf}y_{sup} + x\left(y_{sup} - y_{inf}\right)}{x_{sup} - x_{inf}}$$

donde x son las variables y y son los valores de la función, *sup* quiere decir el inmediato superior (por valor de la variable) en la lista de conocidos y inf el inmediato inferior.

Ejemplo

Nos dan los valores: 1, 5, 2.5, 3 y 8 y que la función $\hbar_{\aleph}$ vale en ellos respectivamente: 0.5, 2, 0.7, 2.1 y -1. Queremos estimar cuánto valdrá en 6.

Vemos que el valor incógnita está comprendido entre dos de los conocidos: 5 y 8, donde la función vale 2 y -1. Aplicando la fórmula para ellos tenemos:

$$\frac{8\times 2 - 5\times(-1) + 6\left(-1-2\right)}{8-5} = \frac{16+5+6(-3)}{3} = \frac{21-18}{3} = 1$$

Resolución

En este caso partimos de que se trata de aplicar la fórmula, y la única complejidad es localizar los elementos inmediato inferior e inmediato superior. ¿En qué caso resultaría esto fácil? ¿Qué pediríamos a los datos para que nos fuese más fácil encontrar estos elementos? Si nos lo planteamos vemos que lo que pediríamos es que nos los diesen ordenados, porque se saca mucho más deprisa entre qué dos está.

Entonces lo que planteamos es:

Leer datos
Ordenarlos por la variable
Localizar entre qué dos posiciones se sitúa el pedido
Aplicar la fórmula

Cada línea podría ser una función independiente.

Por otro lado, los datos son una tabla de valores de una función. Una tabla con dos columnas, o con dos filas: una para el valor de la variable y otra para el valor de la función. Necesitamos pues dos operaciones para tablas: ordenarlas atendiendo a los valores de una de las filas o columnas (pongamos la primera) y localizar la posición de una incógnita en una fila o columna ya ordenada. Como disponemos de plantilla para ordenar tabla por filas (pág. 204), vamos a pensar en una tabla de dos columnas.

Por tanto para ordenar utilizamos la plantilla suministrada, coincidiendo prácticamente con el ejemplo presentado.

En lo que se refiere a buscar la posición del número, podemos usar directamente la idea de búsqueda de un número vista en el correspondiente ejemplo resuelto. Aquí ya damos por hecho que no va a estar, así que paramos la búsqueda cuando la posición izquierda y derecha del intervalo de búsqueda sean contiguas. Esas son las dos posiciones que nos interesan. La notación que usamos para indicar el elemento de la fila fi columna co de la tabla ta es ta[fi][co], por anticipar lo que se va a poner en C.

```
inferior=0
superior=numerodefilas-1
repetir
    centro=(superior+inferior)/2
    compara=tabla[centro][0]
    si compara < pedido entonces
        inferior=centro
    si no
        superior=centro
    final de si
hasta que superior sea igual que inferior+1
```

Al acabar esta función podemos obtener los valores inmediato inferior y superior de las variables en `tabla[inferior][0]` y `tabla[superior][0]` y los de la función en `tabla[inferior][1]` y `tabla[superior][1]`,para aplicar directamente la fórmula.

Escribir la matriz traspuesta

```
1  #include <stdio.h>
2
3  void lee(int filas, int columnas, int original[filas][columnas])
4     {
5     /* Para contar las filas y columnas que se leen */
6     int cuenfil, cuencol;
7     for (cuenfil = 0; cuenfil < filas; cuenfil++
```

```
8           /* Se piden fila a fila */)
9        {
10       printf("Fila %i: ", cuenfil+1);
11       /* Se sacan las filas numeradas
12       a partir de 1 */
13       for (cuencol = 0; cuencol < columnas; cuencol++)
14          {
15          scanf("%i", &original[cuenfil][cuencol]);
16          /* Se leen todas las columnas seguidas */
17          }
18       }
19    }
20
21 void cambia(int filas, int columnas,
22 int original[filas][columnas], int cambiada[columnas][filas])
23    {
24    /* Para contar las filas y columnas que se pasan */
25    int cuenfil, cuencol;
26    for (cuenfil = 0; cuenfil < filas; cuenfil++
27          /* Se barre la matriz original fila a fila */)
28       {
29       for (cuencol = 0; cuencol < columnas; cuencol++)
30          {
31          cambiada[cuencol][cuenfil] = original[cuenfil][cuencol
               ];
32          /* El índice de fila de la original es el de columna
33          de la cambiada y viceversa */
34          }
35       }
36    }
37
38 void escribe(int filas, int columnas,
39 int cambiada[filas][columnas])
40 /* Para evitar errores, en esta función llamamos
41 filas y columnas a lo que son en la cambiada,
42 aunque sean lo contrario en la original */
43    {
44    /* Para contar las filas y columnas que se escriben */
45    int cuenfil, cuencol;
46    for (cuenfil = 0; cuenfil < filas; cuenfil++
47          /* Se escriben fila a fila */)
48       {
49       for (cuencol = 0; cuencol < columnas; cuencol++)
50          {
51          printf(" %i", cambiada[cuenfil][cuencol]);
52          /* Se leen todas las columnas seguidas */
53          }
```

```
54        printf("\n"); /* se cambia de línea tras cada fila */
55        }
56     }
57
58  void main()
59     {
60     int filas, columnas; /* Tamaño de la tabla */
61     printf("¿Filas y columnas de la tabla?\n");
62     scanf("%i %i", &filas, &columnas);
63        {
64        int original[filas][columnas], cambiada[columnas][filas];
65        lee(filas,columnas,original);
66        cambia(filas,columnas,original,cambiada);
67        escribe(columnas, filas, cambiada);
68        }
69     }
```

La lectura se presenta desarrollada. El paso a la matriz traspuesta se hace poniendo cada elemento en la posición con índices de fila y columna intercambiados. La escritura final, también se presenta desarrollada.

Producto de matrices

```
1  #include <stdio.h>
2  #include <listas.h>
3  void calcular(int totfil1, int totcol1, int totcol2,
4  float matriz1[totfil1][totcol1],
5  float matriz2[totcol1][totcol2], float resul[totfil1][totcol2])
6  /* Calcula la matriz producto
7  ENTRADA: filas y columnas de primera, columnas de segunda y
8  matrices operandos
9  SALIDA: matriz producto */
10    {
11    int fila,col,k;
12    /* k se va a usar en el cálculo como
13    columna de la primera y fila de la segunda */
14    for (fila = 0; fila < totfil1; fila++)
15       {
16       for (col = 0; col < totcol2; col++)
17          {
18          resul[fila][col] = 0;
19          for (k=0; k<totcol1; k++)
20             {
21             resul[fila][col] += matriz1[fila][k]*matriz2[k][col
                  ];
22             }
23          }
24       }
```

```
25    }
26 void main()
27    {
28    int totfil1, totcol1, totcol2;
29    printf("Indicar filas y columnas de primera matriz y columnas
          de segunda: ");
30    scanf("%i %i %i", &totfil1, &totcol1,&totcol2);
31       {
32       float matriz1[totfil1][totcol1], matriz2[totcol1][totcol2
            ],
33       resul[totfil1][totcol2];
34       printf("Teclea primera matriz:\n");
35       letanu("%g",matriz1,totfil1,totcol1);
36       printf("Teclea segunda matriz:\n");
37       letanu("%g",matriz2,totcol1,totcol2);
38       calcular(totfil1,totcol1,totcol2,matriz1,matriz2,resul);
39       estanu("%g ",resul,totfil1,totcol2);
40       }
41    }
```

En este caso las lecturas y escrituras utilizan las macros preparadas (pág. 203). El cálculo se hace poniendo a 0 cada elemento y luego sumando en él todos los productos de elementos de la misma fila de la primera matriz y la misma columna de la segunda; el índice de variación del ciclo es la columna de la primera, que coincide con fila de la segunda.

Escribir los puntos más próximos

```
1  /* ENUNCIADO
2  Hacer un programa que lea las coordenadas x-y de una serie de
      puntos en el plano y escriba los n·meros (por el orden en que
       se han leÝdo) de los que estßn mßs pr¾ximos a un centro,
      cuyas coordenadas x-y tambiÚn se leerßn. Se considerarßn
      puntos mßs pr¾ximos el mßs cercano al centro y todos aquellos
       que disten del centro menos del doble que el mßs cercano.
3
4  ESQUEMA DE SOLUCIËN
5  Pedir y leer cußntos puntos hay en el conjunto
6  Pedir y leer sus coordenadas ---- usar funci¾n
7  Pedir y leer las coordenadas del centro
8  ------------------Funci¾n para sacar la distancia del mßs
      cercano al centro
9  Empezar poniendo la distancia mÝnima a un valor enorme
10 Recorrer las filas de la matriz de coordenadas (cada fila es un
      punto)
11          Calcular la distancia de ese punto al centro y ponerlo
               en la posici¾n correspondiente de la lista de
               distancias
12          Si esa distancia es menor que la mÝnima,
```

```
13                La nueva mÝnima es esta
14 ------------------------------------------------------------------------

15 -----------------Funci¾n para escribir todos los que entran en
      la categorÝa de pr¾ximos
16 Recorrer la lista de distancias
17         Si la distancia correspondiente es menor que el doble de
               la mÝnima,
18                Escribir el n·mero de punto
19 ------------------------------------------------------------------------

20 */
21 #include <stdio.h>
22 #include <math.h>
23 #define DIM 2
24
25 void leecor(int npuntos, float coor[npuntos][DIM])
26    {
27    int cuenta;
28    for (cuenta = 0; cuenta < npuntos; cuenta++)
29       {
30       scanf("%g %g",&coor[cuenta][0],&coor[cuenta][1]);
31       }
32    }
33
34 void caldis(int npuntos, float coor[npuntos][DIM], float x,
35 float y, float dist[npuntos], float *dismin)
36    {
37    int cuenta;
38    *dismin= 1e30;
39    for (cuenta = 0; cuenta < npuntos; cuenta++)
40       {
41       dist[cuenta]=sqrt((coor[cuenta][0]-x)*(coor[cuenta][0]-x)+
42       (coor[cuenta][1]-y)*(coor[cuenta][1]-y));
43       if (dist[cuenta] < *dismin)
44          {
45          *dismin = dist[cuenta];
46          }
47       }
48    }
49
50 void escribe(float dismin, int npuntos, float dis[npuntos])
51    {
52    int cuenta;
53    printf("Puntos más próximos: ");
54    for (cuenta = 0; cuenta < npuntos; cuenta++)
55       {
```

```
56          if (dis[cuenta] < 2*dismin)
57             {
58             printf("%i ",cuenta+1);
59             }
60          }
61       }
62
63    void main()
64       {
65       int npuntos;
66       float x,y,dismin;
67       printf("¿Cuántos puntos? ");
68       scanf("%i",&npuntos);
69          {
70          float coord[npuntos][DIM], dist[npuntos];
71          printf("Dar coordenadas de todos los puntos\n");
72          leecor(npuntos,coord);
73          printf("Punto de referencia: ");
74          scanf("%g %g",&x,&y);
75          caldis(npuntos,coord,x,y,dist,&dismin);
76          escribe(dismin,npuntos,dist);
77          }
78       }
```

Al ser puntos en el plano, sus coordenadas se guardan en una tabla de dos columnas. Para calcular las distancias, que se guardan en una lista para uso posterior, se recorre la tabla aplicando la fórmula de distancia euclídea; una vez calculada cada distancia se comprueba si es la mínima, siguiendo el patrón de búsqueda de extremo (pág. 146). Para escribir los resultados se recorre la lista de distancias comprobando si son menores que el doble de la mínima y escribiendo su índice en ese caso.

Calcular la matriz promedio de 4 vecinos

```
1  /* ENUNCIADO
2  Hacer un suavizado de una imagen en escala de grises. La intensidad luminosa de cada pixel
      se leera por pantalla, despues de haber leido las dimensiones. Para el suavizado se
      tomara en cada pixel el promedio de sus vecinos en la imagen original, considerando como
      tales a los situados a la izquierda, la derecha, arriba y abajo. Las filas y columnas
      extremas se dejaran como estaban.
3
4  ESQUEMA DE SOLUCION
5  Pedir y leer la dimensiones
6  Leer los valores de la matriz de luminosidades
7  -----------Funcion para calcular el suavizado
8  Copiar la matriz original en la resultado
9  Recorrer la matriz resultado, desde la segunda fila a la penultima, y lo mismo en columnas
10         Poner en la posicion correspondiente de la matriz resultado el promedio de los
              vecinos, cogiendolos de la matriz original
11 ---------------------------------------------
12 Escribir la matriz resultado
13 */
14 #include <stdio.h>
15 #include <string.h> /* para memcpy */
16 #include <listas.h> /* para lectura y escritura de tablas */
17 #define VECINOS 4
18 void calcular(int totfil, int totcol, float tabla[totfil][totcol], float resul[totfil][
      totcol]) {
19 /* Calcula la matriz promedio de 4 vecinos
20 ENTRADA: filas, columnas y tabla original
21 SALIDA: tabla promedio de 4 vecinos */
22    int f,c;
23    memcpy(resul,tabla,sizeof(float)*totfil*totcol);
24    for (f = 1; f < totfil-1; f++)
25       {
```

```
26        for (c = 1; c < totcol-1; c++)
27           {
28           resul[f][c] = (tabla[f][c-1] + tabla[f-1][c] + tabla[f+1][c] + tabla[f][c+1])/
                VECINOS;
29           }
30        }
31     }
32  void main()   {
33     int totfil, totcol;
34     printf("Indicar filas y columnas: ");
35     scanf("%i %i", &totfil, &totcol);
36        {
37        float tabla[totfil][totcol], resul[totfil][totcol];
38        letanu("%g",tabla,totfil,totcol);
39        calcular(totfil,totcol,tabla,resul);
40        estanu("%g ",resul,totfil,totcol);
41        }
42     }
```

Primero se copia íntegramente la tabla origen sobre la destino utilizando `memcpy`. Después se calcula para cada elemento la fórmula, que es poner el valor que había en la tabla original en las posiciones de fila y columna anterior y posterior. En la figura 0.14 tienes una animación/-presentación que explica la construcción del programa, partiendo de lo básico ya conocido de leer y escribir tablas. Requiere que abras el PDF con el visor de Adobe. Pulsando en ella vas avanzando.

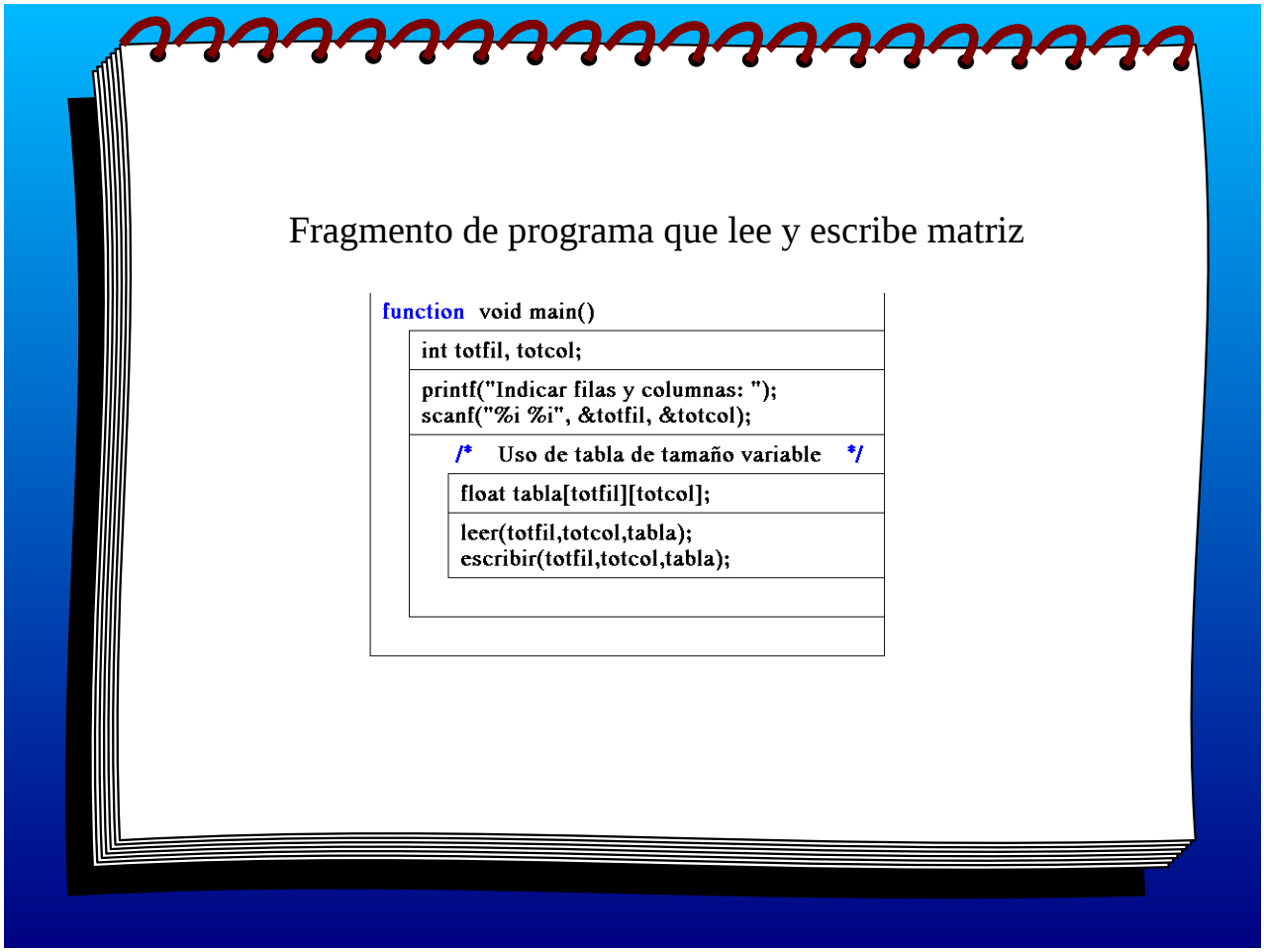


Figura 0.14: Cómo se llega al programa que calcula el promedio de los cuatro vecinos

Importes de alquileres

```
1 /* ENUNCIADO
2 Programa que calcule lo que tiene que pagar cada persona en
    concepto de alquiler de locales. El programa leera la
    cantidad de locales que hay y la cantidad de arrendatarios.
    No tienen por que coincidir, porque una persona puede tener
    varios locales. A continuacion leera el nombre de cada
    arrendatario y el precio de cada local, asi como el
    arrendatario que lo tiene. El arrendatario se indicara por su
     numero, correspondiente a la posicion en que se ha leido su
```

```
   nombre. Finalmente escribira el nombre de cada arrendatario y
    el importe total que tiene que pagar.
3
4  ESQUEMA DE SOLUCION
5  Pedir y leer numero de locales y de arrendatarios
6  Pedir y leer los nombres de los arrendatarios en lista -------
      usar funcion
7  Pedir y leer precio y arrendatario de cada local, en sendas
      listas ------- usar funcion
8  ------------------------Funcion para calcular importes a pagar
9  Poner la lista de facturas de arrendatarios a 0
10 Recorrer los locales
11         A la factura dada por el numero de arrendatario
              correspondiente sumarle el precio correspondiente
12 Recorrer los arrendatarios
13         Escribir el nombre y la factura correspondiente
14 ------------------------------------------------------------
15 */
16 #include <stdio.h>
17 #include <string.h>
18 typedef float Euros;
19 #define NOMAX 41
20
21 void leeloc(int nloc, Euros precio[nloc], int arren[nloc])
22    {
23    int cuenta;
24    for (cuenta = 0; cuenta < nloc; cuenta++)
25       {
26       scanf("%g %i",&precio[cuenta],&arren[cuenta]);
27       }
28    }
29
30 void leen(int nper, char nombres[nper][NOMAX])
31    {
32    int cuenta;
33    for (cuenta = 0; cuenta < nper; cuenta++)
34       {
35       scanf("\n%40[^\n]",nombres[cuenta]);
36       }
37    }
38
39 void cales(int nloc, Euros precio[nloc], int arren[nloc], int
      nper,
40 char nombres[nper][NOMAX])
41    {
42    int cuenta;
43    Euros factura[nper];
```

```
44    memset(factura,0,nper*4);
45    for (cuenta = 0; cuenta < nloc; cuenta++)
46       {
47       factura[arren[cuenta]-1] += precio[cuenta];
48       }
49    for (cuenta = 0; cuenta < nper; cuenta++)
50       {
51       printf("%s debe pagar %g\n",nombres[cuenta],factura[cuenta
            ]);
52       }
53    }
54
55 void main()
56    {
57    int nloc, nper;
58    printf("Cuntos locales? ");
59    scanf("%i",&nloc);
60    printf("Cuntos arrendatarios? ");
61    scanf("%i",&nper);
62       {
63       Euros precio[nloc];
64       int arren[nloc];
65       char nombres[nper][NOMAX];
66       printf("Dar nombres y apellidos de cada arrendatario\n");
67       leen(nper,nombres);
68       printf("Dar precio y numero de arrendatario de cada local\
            n");
69       leeloc(nloc,precio,arren);
70       cales(nloc,precio,arren,nper,nombres);
71       }
72    }
```

Se lee primero la lista de nombres de arrendatarios; como puede haber espacios separando apellidos, se usa el formato `%[^\n]`. Después se lee de cada local su precio y el arrendatario, identificado por su número en la lista de nombres. En el cálculo, primero se pone a 0 toda la lista de facturas usando `memset`. Luego se suma cada precio a la factura del arrendatario correspondiente; obsérvese como para indicar un índice puede usarse cualquier expresión, incluyendo elementos de listas. Finalmente se recorren simultáneamente las listas de nombres y facturas, escribiendo sus valores.

Bibliografía y recursos de programación.

Bibliografía básica

[Got05] B. Gottfried. Programación en C. McGraw-Hill, 2005.

Es un buen libro para aprender el lenguaje C. Enfoca cada tema con muchos ejemplos y programas analizados detalladamente. Cada capítulo contiene preguntas de repaso, problemas para desarrollar manualmente y problemas para programar. Esto último lo convierte en un libro recomendable porque el aprendizaje se basa en la práctica del lenguaje. Contiene apéndices donde se resume la sintaxis del lenguaje.

[Joy05] L. Joyanes Aguilar, I. Zahonero Martínez. Programación en C. Metodología, algoritmos y estructuras de Datos. McGraw-Hill, 2005.

Este libro es una revisión de su misma obra del año anterior en el que se han incluido nuevos temas como son la recursividad, los algoritmos de ordenación y búsqueda y los tipos de datos abstractos (TAD/Objetos). La obra ofrece muchos ejercicios resueltos y propuestos.

[Joy04] L. Joyanes Aguilar, I. Zahonero Martínez. Algoritmos y Estructuras de Datos, una perspectiva en C. McGraw-Hill, 2004.

Este libro sigue los programas clásicos de las disciplinas Estructuras de Datos y de la Información. En sus primeros capítulos introduce el concepto de algoritmo y las distintas formas de medir su complejidad y eficiencia. Asimismo describe de un modo sencillo y didáctico las estructuras de datos más simples tales como los arrays y estructuras para ir introduciéndose en otras más complejas como listas, pilas, colas, grafos y árboles. También dedica unos capítulos al diseño de algoritmos de búsqueda, ordenación y recursividad. El código está disponible en la Web.

[Gar02] F. García, J. Carretero, J. Fernández, A. Calderón. El lenguaje de programación C. Diseño e implementación de programas. Prentice-Hall. 2002.

Es un libro que está pensado como un texto general de diseño e implementación de aplicaciones usando el lenguaje C. Cubre tanto los aspectos básicos de programación como los más avanzados del lenguaje, incluyendo las características del estándar C99. Además introduce consejos de programación que enfatizan los aspectos problemáticos del lenguaje C y cómo hacer frente a estos; advertencias que previenen un mal uso del lenguaje o la codificación de programas con bajo rendimiento y una lista de los errores más frecuentes al término de cada capítulo.

Repositorios de programas resueltos en Internet

Bastantes están en inglés. Es útil saber inglés, pero quien no sepa puede recurrir a un traductor automático; aunque no lo hacen bien, sí suele ser suficiente para entender de qué va.

`http://www.codechef.com/problems/easy` A nuestro nivel sólo la categoría easy. Mirar que las soluciones estén en C. Comprobar que no aparecen elementos de C que no hayamos dado

`http://www.hpcodewars.org/index.php?page=samples` Entrando en cada año a partir de 2005, que es cuando hay soluciones. Entrar a cada enunciado, empezando por arriba. Si viene nivel considerar sólo los que ponga "Novice" o "Novice/Advanced", aunque típicamente sólo los primeros de esta categoría son de un nivel suficientemente bajo. Si sólo vienen puntos, no pasar de los de 6 puntos. Comprobar que las soluciones están en C

Libro electrónico: https://ebuc.unican.es/Record/Xebook1-2365 `http://foro.elhacker.net/ejercicios/ejercicios_resueltos_c_programacion_estructurada-t201980.0.html`

`http://www.programmingsimplified.com/c-program-examples` Abarca desde lo muy sencillo hasta por encima del nivel de este curso. Lo más interesante está a medio camino.

Por último un repositorio de problemas donde no están resueltos, pero hay verdaderamente muchos: `http://poj.org/problemlist` En seguida supera a la asignatura, pero en tanta cantidad, los hay que valen.

Bibliografía complementaria

[Bra97] G. Brassard y P. Bratley. Fundamentos de algoritmia. Prentice-Hall, 1997.

Este libro contiene un estudio detallado y sistemático del diseño y análisis de algoritmos, proporcionando al alumno las herramientas básicas para el desarrollo de sus propios algoritmos. Se desarrollan técnicas para algoritmos voraces, del tipo divide y vencerás, programación dinámica y técnicas de grafos. Se presentan las técnicas de forma general y se muestran ejemplos de aplicación. Emplea seudocódigo para representar los algoritmos.

[Bow94] C. Bowman. Algorithms and Data Structures. An approach in C. Saunders College Publishing, 1994.

Es un libro que introduce al lector en el análisis y desarrollo de algoritmos. Es bastante claro y conciso. Enfatiza el concepto de tipo de dato abstracto y usa el lenguaje C para implementar los algoritmos. Recoge las estructuras de datos y algoritmos de uso más general.

[Ker91] B. Kernighan y D. Richie. El lenguaje de programación C. Prentice-Hall, 1991.

Es un clásico, escrito por los creadores del lenguaje. De hecho esta versión del libro constituye la referencia del ANSI C. Está escrito para programadores con experiencia. Tiene la ventaja de usar un lenguaje directo y cubre en relativamente pocas páginas toda la sintaxis del C. Desde un punto de vista didáctico no es muy recomendable pero constituye una buena referencia cuando ya se tiene conocimiento del lenguaje.

[Ral97] A. Ralston y H. Neill. Algorithms. NTC Publishing Group, 1997.

Es un libro de contenido básico presentado de forma concisa, clara y simple. Usa seudocódigo para desarrollar la solución de todos los problemas. También contiene una breve introducción a la verificación de algoritmos. El formato de guía de bolsillo es una ventaja adicional.

Direcciones Web

1. La Web de Programador. http://www.lawebdelprogramador.com/
   Portal que ofrece diferentes recursos relacionados con la informática y la programación. Tiene algunas referencias al C en https://www.lawebdelprogramador.com/cursos/C-Visual-C/index1.html

2. Ingeniería de programación (en inglés). http://www.rspa.com/
   Sitio Web mantenido por R.S. Pressman & Associates que ofrece más de 1000 recursos de ingeniería del software abordando los diferentes tópicos de la disciplina.

3. The Stony Brook Algorithm Repository (en inglés). https://algorist.com/algorist.html
   Portal Web que recoge una lista amplia de implementación de algoritmos.

4. Otro curso de C: https://www.tutorialesprogramacionya.com/cya/
   Llega a más de nuestro curso, ya que se plantea la posibilidades del C al completo.

Índice alfabético