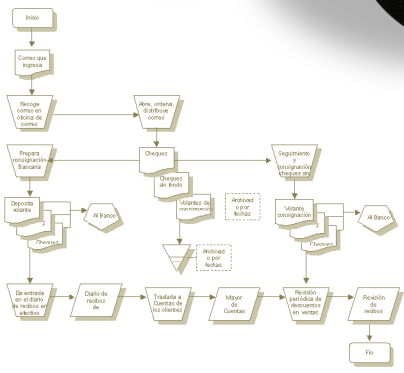
PROBLEMAS RESUELTOS EN LENGUAJE C

$$\text{varianza} = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - \frac{1}{n^2}\left(\sum_{i=1}^{n} x_i\right)^2$$

$$\text{media} = \frac{1}{n}\sum_{i=1}^{n} x_i$$



Ramón Mollá Vayá
Inmaculada García García
Laura Sebastiá Tarín
Jon Ander Gómez Adrián

José Miguel Alonso Ábalos
David Guerrero López
Miguel Ángel Martín Caro

EDITORIAL
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Ramón Mollá Vayá
Inmaculada García García
Laura Sebastiá Tarín
Jon Ander Gómez Adrián
José Miguel Alonso Ábalos
David Guerrero López
Miguel Ángel Martín Caro

PROBLEMAS RESUELTOS EN LENGUAJE C

EDITORIAL
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Primera edición, 2005 ▪ reimpresión, 2016

© de la presente edición:
Editorial Universitat Politècnica de València
www.editorial.upv.es

Venta: www.lalibreria.upv.es

© Ramón Mollá Vayá
Inmaculada García García
Laura Sebastiá Tarín
Jon Ander Gómez Adrián
José Miguel Alonso Ábalos
David Guerrero López
Miguel Ángel Martín Caro

ISBN: 978-84-9705-883-4 (versión impresa)
ISBN: 978-84-1396-148-4 (versión electrónica)
Ref. editorial: 6648_01_01_01

Queda prohibida la reproducción, distribución, comercialización, transformación, y en general, cualquier otra forma de explotación, por cualquier procedimiento, de todo o parte de los contenidos de esta obra sin autorización expresa y por escrito de sus autores.

Resumen

Este libro va destinado a todos aquellos estudiantes de primeros cursos que reciben formación en programación de computadores. Concretamente se ha empleado el lenguaje C, ampliamente extendido en el mundo profesional. El objetivo de este libro no es recopilar una colección de ejercicios sin más, sino adaptarlos al nivel básico de introducción a la programación. Se agrupan por capítulos, de forma que es fácil para el alumno practicar sólo un aspecto concreto del lenguaje. Cada capítulo consta de entre 25 y 30 ejercicios con tres niveles de dificultad. También aparecen al menos 3 ejercicios completamente analizados y diseñados, 3 sólo analizados y el resto simplemente propuestos. Todos están completamente resueltos y clasificados en el código QR que aparece al final del índice. También hay un tema final de ejercicios que aglutinan todos los temas tocados y otro de sugerencias de corrección de errores frecuentes cometidos por principiantes.

ÍNDICE

FICHEROS COMPLEMENTARIOS

Al descargar este enlace, encontrará dos carpetas: ejercicios resueltos y compiladores gratuitos. Cada carpeta contiene diferentes archivos y enlaces a software que complementan el texto del libro.

Peso archivo comprimido 3M

http://tiny.cc/0683_Mat_complementario

Introducción

Este libro va destinado a todos aquellos estudiantes de primeros cursos de ingeniería en sus diferentes vertientes, incluyendo la ingeniería en informática, que reciben formación básica en programación de computadores en lenguaje C. No obstante, debido al nivel del libro, perfectamente podría ser utilizado también en academias, en cursos de bachiller o formación profesional en los que se trabaje también con el lenguaje C, ampliamente extendido en el mundo profesional.

El objetivo de este libro no es recopilar una colección de ejercicios sin más, sino adaptarlos al nivel básico de introducción a la programación, agrupándolos en función de diferentes aspectos del lenguaje, de forma que sea fácil para el alumno practicar principalmente un aspecto concreto del lenguaje en cada capítulo. Todos los ejercicios son originales de sus autores, por lo que el lector podrá obtener de este libro una ayuda complementaria en español a toda la amplia bibliografía y ejercicios que puede obtener por otras vías, como por ejemplo, en Internet.

Cada capítulo incluye alrededor de unos 30 ejercicios con diferentes niveles de dificultad. También aparecen al menos 3 ejercicios completamente analizados y diseñados, al menos 3 ejercicios sólo analizados y el resto simplemente propuestos. Todos están completamente resueltos y clasificados en el QR. También hay un capítulo final de ejercicios que aglutinan a todos los demás incluidos en el libro y otro de sugerencias de corrección de errores frecuentes cometidos por principiantes.

En la presente edición no se ha incluido nada referente a programación dinámica, reserva de memoria dinámica, ficheros avanzados, llamadas al sistema operativo y demás conceptos que exceden el nivel básico introductorio del lenguaje. En cambio, sí que se incluyen todas las sentencias de control de flujo, los operadores básicos, funciones, estructuras de datos homogéneos (cadenas, vectores y matrices) y heterogéneos (estructuras).

Se ha intentado, en la medida de lo posible, no desarrollar programas que sean dependientes del sistema operativo, de una versión dada o de un fabricante en concreto. No obstante, en la práctica esto no es del todo posible, por lo que en algunos casos, los ejercicios tienen alguna instrucción que no ha habido más remedio que incluir para que los programas funcionen correctamente en el compilador gratuito DEV-C++ sobre el sistema operativo MS-Windows. En este sentido, la versión que aparece en el libro, está libre de sentencias dependientes del entorno de programación o del operativo, tal y como se debería de ejecutar en un entorno tipo ANSI C. Sin embargo, la versión que aparece en el QR contiene algunas instrucciones específicas que garantizan la correcta ejecución del programa en el entorno del DEV C++ citado anteriormente. Este hecho no altera el funcionamiento del programa, ni el algoritmo utilizado. Tan sólo lo adapta al entorno para que pueda ejecutarse tal y como debería haberlo hecho en el libro.

Todos los programas han sido compilados y ejecutados en el entorno de programación anteriormente indicado. Todos funcionan correctamente y no hay, a fecha de publicación, ninguna errata que impida a ninguna solución poder ser

compilada y ejecutada sin errores. Para poder verificar este punto, se ha incluido para cada solución, su correspondiente programa ejecutable, para que el lector pueda comprobar este punto, observar cómo funciona la solución ofrecida y hacerse una idea de qué se pide en el enunciado.

Estos programas no emplean las mejores soluciones conocidas hasta el momento. En algunos casos, existen versiones más rápidas o eficientes que la solución ofertada. Dado que el curso al que va dedicada esta publicación suele ser primer curso de programación, el objetivo del libro no es la eficiencia, que se deja para cursos superiores, sino el desarrollar buenos hábitos de programación, buenas técnicas y metodologías de desarrollo de código fuente, primando la pedagogía sobre la eficiencia de código.

Los ficheros que contienen las soluciones en el QR que se acompaña están organizados por directorios que coinciden con los nombres de los capítulos del libro. Dentro de cada directorio aparecen diferentes ficheros acabados en ".c" (punto c minúscula). El nombre del fichero hace referencia al número del ejercicio propuesto en el capítulo indicado. Así mismo, también han sido incluidos los ficheros auxiliares de datos y las cabeceras necesarios para la correcta ejecución del programa.

En algunos casos, existen varias versiones (soluciones) del mismo ejercicio. En esta situación, se ha incluido un segundo fichero con el mismo nombre y extensión, pero que presenta un sufijo -2 o -3 al nombre. El lector observará que en cada capítulo, el código fuente existente sigue diferentes pautas o normativas de escritura. Por ejemplo, algunos autores prefieren incluir la primera sentencia de un bloque en la misma línea en la que se abre la llave, mientras que otros lo hacen en la siguiente. Unos no indentan el código si no hace falta, mientras que otros lo hacen con más insistencia. En realidad, afortunadamente, no existe una normativa única, por lo que cada cual tiene plena libertad para adoptar la que más le guste. Deliberadamente no se ha seguido ninguna normativa en concreto que dé uniformidad al código precisamente para que el programador novel no quede "viciado" por una en particular y crea que sólo se puede programar de una determinada forma. Así, también se facilita al lector ampliar sus conocimientos, permitiéndole entender mejor cualquier tipo de código al que pudiera acceder, sea de este libro o no.

Los autores del presente libro hemos realizado esta publicación dedicándoles incontables horas de fatigas. Se han realizado varias versiones revisadas y varios ciclos de depuración de errores y erratas. Hemos intentado hacerlo lo mejor que sabíamos. Sin duda alguna que existe una gran cantidad de problemas genéricos y específicos de las diferentes ramas de la ingeniería que podrían haber sido incluidos también en esta publicación. No obstante, esta situación habría reducido la universalidad de nuestra intención y el rango del público objetivo. Por otro lado, habría alargado aún más el tamaño de esta publicación, haciendo su adquisición más onerosa para el lector.

Esperando haber podido cumplir con nuestro objetivo.

Los autores

Capítulo 1
Entrada y salida

1. Introducción

Este capítulo contiene problemas sobre los aspectos más básicos del lenguaje C:

- Instrucciones de entrada y salida (`printf` y `scanf`).
- Expresiones de diferente tipo: numéricas, relacionales, lógicas y de caracteres.
- Tipos de datos básicos: `int, float, char.`
- Manejo de variables.
- Directivas de preprocesador sencillas (`#define`).

2. Problemas Analizados y Resueltos

1. Indicar cuál será el resultado de la ejecución de este programa:

```
#include <stdio.h>
#define DATO 4
main()
{
int x, y=1;
float z;
x=3/2+10-3;
z=3.0/2.0+10-3;
printf ("%d -- %f\n", x, z);
printf ("%d\n", DATO*3+1/2);
printf ("%d\n", 5+2-1>5);
printf ("%d\n", 3>1 || DATO*2<1);
printf ("%d\n", 5!=1 && DATO>2);
y=x;
printf ("%d\n", x+y<=z);
}
```

Solución

Este problema consiste básicamente en la evaluación de expresiones aritméticas, relacionales y lógicas. Para resolver este problema, se debe seguir paso a paso cada una de las instrucciones que componen el programa:

#include <stdio.h>	Como el programa utiliza la instrucción `printf`, se debe incluir esta biblioteca.
#define DATO 4	Define la constante `DATO` con el valor 4.
main() {	Marca de inicio del programa.
int x, y=1; float z;	Declaración de las variables `x, y` (enteras) y `z` (`flota`). De esta forma, se indica que las variables `x, y` pueden almacenar valores enteros y `z` puede contener valores reales. Además, también se asigna a la variable `y` el valor 1.

x=3/2+10-3;	Con esta instrucción, se asigna a la variable `x` el resultado de la expresión a la derecha del signo igual. Según la precedencia de los operadores, se debe evaluar la división en primer lugar. Como los operandos son números enteros, se calcula la división entera, luego el resultado será 1. Así, la expresión resultante es: `1+10-3`, que se evalúa de izquierda a derecha al tener todos los operadores la misma precedencia. El resultado final es 8, que se asigna a `x`.
z=3.0/2.0+10-3;	Esta instrucción es similar a la anterior. La principal diferencia está en la división que aparece en la expresión, ya que los operandos son números reales y por tanto, se efectúa la división real. Así, la expresión resultante es: `1.5+10-3`, cuyo resultado final es 8.5, que se asigna a `z`.
printf ("%d -- %f\n", x, z);	Esta instrucción simplemente imprime el contenido de las variables `x` y `z`. En pantalla aparecerá: `8 — 8.500000`
printf ("%d\n", DATO*3+1/2);	Esta instrucción imprime el resultado de la expresión que aparece fuera de las comillas. La evaluación de esta expresión se realiza siguiendo la precedencia de los operadores. Como el producto y la división tienen la misma prioridad, se evalúan de izquierda a derecha. Teniendo en cuenta que el valor de la constante `DATO` es 4, se obtiene: `12+0` (división entera). Por tanto, en pantalla aparecerá `12`.
printf ("%d\n", 5+2-1>5);	Al igual que en la anterior, esta instrucción imprime el resultado de la expresión. En este caso, se debe evaluar en primer lugar la expresión aritmética que aparece a la izquierda del >, cuyo resultado es 6. Ahora se debe evaluar la expresión relacional resultante: 6>5, que es verdadera. Por tanto, en pantalla aparecerá un 1, que es el valor que C utiliza internamente para representar el valor verdadero. Se utiliza 0 para representar el valor falso.
printf ("%d\n", 3>1 || DATO*2<1);	Como en los casos anteriores, en primer lugar se debe evaluar la expresión. Ahora se trata de una expresión lógica. Se evalúan los operandos de esta expresión: `3>1`, que es verdadero y `DATO*2<1`, que es falso. Como la expresión lógica

	es OR, y al menos uno de los operandos es verdadero, la expresión será verdadera y por tanto, se imprimirá un 1.
printf ("%d\n", 5!=1 && DATO>2);	En primer lugar, se evalúan los operandos de la expresión lógica: `5!=1`, que es verdadero y `DATO>2`, que también es verdadero. Al tratarse de una expresión lógica AND y ser ambos operandos verdaderos, la expresión será verdadera y por tanto, se imprimirá un 1.
y=x;	Esta instrucción es una asignación entre dos variables. La asignación siempre se realiza de derecha a izquierda, es decir, en este caso `y` tomará como valor el contenido de `x`, que es 8. Así que ahora, tanto `x` como `y` valen 8.
printf ("%d\n", x+y<=z);	Al evaluar esta expresión lógica, se deben sustituir las variables por su contenido, con lo que se obtiene: `8+8<=8.500000` Esta expresión es falsa, por lo que en pantalla se imprimirá un 0.
}	Marca de fin de programa.

2. Este programa muestra el resultado de la ecuación $a = b^2 + 1$ para distintos valores de b (0, 2 y –2). ¿Es correcto?. Si no lo es, escribir el programa correctamente.

```
#include <stdio.h>
main()
{
int a, b=0;
a=b*b+1;
printf ("Si b=%d, entonces a=%d\n", b, a);
b=2;
printf ("Si b=%d, entonces a=%d\n", b, a);
b=-2;
printf ("Si b=%d, entonces a=%d\n", b, a);
}
```

Solución

Para saber si este programa realiza correctamente la tarea para la que se ha implementado, hay dos posibilidades: (1) escribirlo en el ordenador y ejecutarlo, (2) hacer la traza.

Si se escoge realizar la traza, se debe seguir cada una de las instrucciones, determinando en cada caso el nuevo valor de las variables y los mensajes que aparecen por pantalla.

<table>
<tr><td><pre>#include <stdio.h>
main()
{</pre></td><td>Inicio del programa.</td></tr>
<tr><td><pre>int a, b=0;</pre></td><td>Declaración de variables.<table><tr><td>Variable</td><td>Contenido</td></tr><tr><td>a</td><td>indeterminado</td></tr><tr><td>b</td><td>0</td></tr></table></td></tr>
<tr><td><pre>a=b*b+1;</pre></td><td>Se evalúa la expresión a la derecha del igual y el resultado se asigna a la variable a. Por tanto: 0*0+1=1.<table><tr><td>Variable</td><td>Contenido</td></tr><tr><td>a</td><td>1</td></tr><tr><td>b</td><td>0</td></tr></table>Como se puede observar, la variable a almacena el resultado de la expresión, no la expresión en sí, de manera que, aunque cambie el valor de b, a mantendrá el valor asignado.</td></tr>
<tr><td><pre>printf ("Si b=%d, entonces a=%d\n", b, a);</pre></td><td>Se imprime en pantalla el contenido de las variables b y a. El mensaje que aparece es:
<pre>Si b=0, entonces a=1</pre>Hasta ahora el programa es correcto, ya que $a = b^2 + 1$ para b=0, resulta a=1.</td></tr>
<tr><td><pre>b=2;</pre></td><td>Se asigna el valor 2 a b:<table><tr><td>Variable</td><td>Contenido</td></tr><tr><td>a</td><td>1</td></tr><tr><td>b</td><td>2</td></tr></table></td></tr>
</table>

	Esta nueva asignación no implica que `a` cambie de valor, como ya se ha explicado. Si se desea que varíe el contenido de `a`, se debe especificar con una nueva instrucción.
`printf ("Si b=%d, entonces a=%d\n", b, a);`	El mensaje que aparece es: `Si b=2, entonces a=1` Esto es incorrecto, ya que $a = b^2 + 1$ para b=2, resulta a=5.

Ya no es necesario seguir con la traza porque se ha comprobado que el programa es incorrecto, es decir, no realiza correctamente la tarea para la cual se ha implementado.

El problema que se observa es que cuando varía el valor de la variable `b`, la variable `a` no cambia de acuerdo con este valor. Ya se ha comentado que para que una variable cambie de valor es necesario escribir una instrucción de asignación (o bien leer un valor con `scanf`). Es un error pensar que si se asigna a la variable `a` una expresión en función de otra variable `b`, cuando cambie el valor de `b` automáticamente cambiará el valor de `a`. El programa correcto deberá incluir una asignación sobre la variable `a` para cada cambio de la variable `b`:

```
#include <stdio.h>
main()
{
int a, b=0;
a=b*b+1;
printf ("Si b=%d, entonces a=%d\n", b, a);
b=2;
a=b*b+1;
printf ("Si b=%d, entonces a=%d\n", b, a);
b=-2;
a=b*b+1;
printf ("Si b=%d, entonces a=%d\n", b, a);
}
```

Si se ejecuta este programa, la salida por pantalla que se obtiene es:

```
Si b=0, entonces a=1
Si b=2, entonces a=5
Si b=-2, entonces a=5
```

3. Una empresa de venta de recambios de automóviles necesita un programa que calcule y muestre el precio final en euros de un producto. Para ello, se debe aplicar la siguiente fórmula:

$$\text{precio\_neto} = \text{precio\_coste} * \frac{100 + \text{margen}}{100}$$

   El precio de coste en euros y el margen en tanto por ciento que desea obtener la empresa para el producto se introducirán por teclado.

Análisis

Una de las aplicaciones principales de un programa informático es la automatización de un proceso que transforma unos datos de entrada en unos datos de salida. Este proceso puede ser de muchos tipos: cálculos más o menos complejos, clasificaciones, ordenaciones, etc.

Cuando se va a utilizar un programa para automatizar un proceso, la primera actividad a realizar debe ser determinar cuáles son los datos de entrada al programa y los datos de salida que éste debe producir. En el caso que se va a resolver, se observa que los datos de entrada serán el precio de coste en euros y el margen que la empresa desea obtener, ya que a partir de estos datos, se calculará el precio neto, que es precisamente el dato de salida que el programa debe obtener.

En segundo lugar, se debe identificar el proceso que el programa debe realizar para transformar los datos de entrada en los datos de salida. En este problema, el enunciado ya indica la fórmula que se debe aplicar para obtener el precio neto.

Resumiendo:

1. datos de entrada: precio de coste y margen
2. datos de salida: precio neto
3. proceso: aplicar la fórmula $\text{precio\_neto} = \text{precio\_coste} * \frac{100 + \text{margen}}{100}$

Ahora se debe plasmar esto en un programa. Las tareas básicas que un programa debe realizar son:

1. Leer los datos de entrada y almacenarlos en variables.
2. Obtener los datos de salida, que se almacenarán en otras variables, mediante la aplicación del proceso indicado sobre los datos de entrada.
3. Escribir por pantalla los datos de salida.

Así, el programa que resuelva este problema deberá:

1. Leer el precio de coste y el margen y almacenarlos en variables, que se llamarán, por ejemplo, `precio_coste` y `margen`.
2. Obtener el precio neto, almacenándolo en la variable `precio_neto`, mediante la aplicación de la fórmula:

$$\text{precio\_neto} = \text{precio\_coste} * \frac{100 + \text{margen}}{100}.$$

3. Escribir por pantalla el contenido de la variable `precio_neto`.

El siguiente paso será determinar el tipo de las variables que se van a utilizar a partir de los datos que van a almacenar. En primer lugar, se decide de qué tipo serán las variables que van a almacenar datos de entrada ya que, en muchas ocasiones, éstas determinan el tipo del resto de variables. Así, la variable `precio_coste` almacenará datos reales ya que este precio se debe introducir en euros, por tanto, será de tipo `float`. En cuanto a la variable `margen`, el enunciado no especifica si el margen será un valor entero o con decimales, por lo que para permitir un uso más amplio del programa, se asume que puede contener decimales y por tanto también será de tipo `float`. Una vez determinado el tipo de las variables de entrada, si se observa el proceso que se va a realizar sobre ellas, se puede concluir que la variable de salida `precio_neto` también será de tipo `float`, ya que la fórmula anterior aplicada sobre datos reales producirá un dato real.

Por último, se debe estudiar detenidamente el proceso que debe realizar el programa. En este caso, ya se ha comentado que únicamente se debe aplicar la fórmula $\text{precio\_neto} = \text{precio\_coste} * \frac{100 + \text{margen}}{100}$. Esta fórmula se puede escribir en C de diferentes maneras, por ejemplo:

```
precio_neto=precio_coste*(100+margen)/100
precio_neto=precio_coste/100*(100+margen)
```

Es importante fijarse en la precedencia de los operadores ya que si se escribe la fórmula anterior como:

```
precio_neto=precio_coste*100+margen/100
```

no producirá el mismo resultado.

Solamente quedaría añadir las instrucciones necesarias para la lectura de los datos de entrada (`scanf`) y para la impresión en pantalla de los datos de salida (`printf`).

Implementación

```
#include <stdio.h>

/* Programa que calcula el precio neto de un producto
   de una empresa de venta de recambios de automoviles */

main()
{
float precio_neto, precio_coste, margen;

/* Entrada de datos: precio_coste y margen */
printf ("Cual es el precio de coste?");
scanf ("%f", &precio_coste);

printf ("Cual es el margen?");
scanf ("%f", &margen);

/* Proceso: calculo del precio neto */

precio_neto=precio_coste*(100+margen)/100;

/* Salida de datos */
printf ("El precio neto de este producto es de %f euros\n",
precio_neto);
}
```

4. Escribir un programa que lea un carácter en minúsculas e imprima en pantalla el mismo carácter en mayúsculas.

Análisis

Al igual que en el problema anterior, se debe determinar en primer lugar, cuáles son los datos de entrada que recibirá el programa y cuáles los datos de salida que deberá producir. En este caso, hay un único dato de entrada, un carácter en minúscula, que el proceso en cuestión debe transformar en la mayúscula correspondiente, que es el dato de salida. Así:

1. datos de entrada: carácter en minúscula
2. datos de salida: el mismo carácter en mayúscula
3. proceso: transformar el carácter en minúscula al correspondiente carácter en mayúscula.

Por tanto, el programa deberá:

1. Leer un carácter en minúscula que se almacenará en la variable `minuscula`.
2. Transformar este carácter en minúscula al correspondiente carácter en mayúscula, que se almacenará en la variable `mayuscula`.
3. Escribir en pantalla el carácter en mayúscula obtenido, es decir, el contenido de la variable `mayuscula`.

En este punto, es necesario determinar el tipo de las variables de entrada y de salida. En este caso, ya que almacenarán caracteres, tanto `mayuscula` como `minuscula` deben ser declaradas de tipo `char`, que es el único que ofrece el lenguaje C para almacenar este tipo de datos.

Por último, se debe precisar cuál será el proceso que permitirá transformar un carácter en minúsculas al correspondiente en mayúsculas. Para ello, se utiliza el código ASCII del carácter en minúscula. Cada uno de los caracteres que es posible representar en el ordenador ('1', '2', 'a', 'b', 'A', 'B', '?', etc.) tiene asociado un código que lo identifica, llamado código ASCII. Así, el carácter '1' tiene el código 49, '2' tiene el código 50, 'a' tiene el código 97, 'b' el 98, 'A' el 65, 'B' el 66, '?' el 63, etc. Estos códigos se han establecido mediante un estándar internacional y todos los ordenadores utilizan el mismo código para identificar al mismo carácter. Es posible conocer el código ASCII de un carácter con el siguiente programa, sustituyendo 'a' por el carácter en cuestión:

```
#include <stdio.h>
main()
{
printf ("%d\n", 'a');
}
```

Esto es posible porque internamente el lenguaje C representa un carácter mediante este código, es decir, si se intenta imprimir un carácter como un número entero como en el programa anterior, se escribe su código ASCII. De igual manera, si se almacena un carácter en una variable de tipo `char`, el contenido de la variable es precisamente su código ASCII.

Para la resolución de este problema es necesario conocer cómo están asignados los códigos ASCII a cada uno de los caracteres más importantes:

1. los caracteres que representan dígitos (0, 1, 2, ..., 9) tienen asignados códigos consecutivos entre el 48 (para el carácter '0') y el 57 (para el carácter '9')
2. los caracteres que representan letras del abecedario en mayúscula (A, B, ..., Z) tienen asignados códigos consecutivos entre el 65 (para el carácter 'A') y el 90 (para el carácter 'Z')
3. los caracteres que representan letras del abecedario en minúscula (a, b, ..., z) tienen asignados códigos consecutivos entre el 97 (para el carácter 'a') y el 122 (para el carácter 'z').

Se observa que el código asignado al carácter 'a' es el código asignado al carácter 'A' más 32. Lo mismo ocurre para los caracteres 'b' y 'B', y así sucesivamente hasta los caracteres 'z' y 'Z'. Por tanto, es posible determinar el código ASCII de un carácter en mayúscula a partir del código ASCII del mismo carácter en minúscula y restándole 32. Este es el proceso que se debe implementar en el programa que resuelva el problema que nos ocupa.

Como se ha comentado, cuando se almacena un carácter en una variable de tipo `char`, el contenido de la variable es precisamente el código ASCII de este carácter. Por tanto, es posible obtener el código ASCII del carácter en mayúsculas restando 32 a la variable que contiene el carácter en minúsculas:

```
mayuscula = minuscula – 32
```

Con esta instrucción se obtiene en la variable `mayuscula` el código ASCII correspondiente al carácter en minúscula, que puede imprimirse como carácter para obtener el carácter en mayúscula.

Implementación

```
#include <stdio.h>

/* Programa que lee un caracter en minusculas y lo escribe en
   mayusculas */
main()
{
char minuscula, mayuscula;
```

```
/* Lectura de datos: minuscula. */

printf ("Introduce un caracter en minuscula: ");
scanf ("%c", &minuscula);

/* Proceso: transformacion a mayuscula */

mayuscula = minuscula - 32;

/* Salida */

printf ("El caracter %c en mayuscula es %c\n", minuscula,
mayuscula);
}
```

Otra opción para realizar esta conversión sería utilizar la siguiente instrucción:

```
mayuscula = minuscula – 'a' + 'A';
```

De este modo, no es necesario recordar el orden en el que los diferentes caracteres aparecen en la tabla ASCII.

Nota: No es posible (por el momento) asegurar que el carácter introducido está en minúscula, por lo que se supondrá que el usuario lo hace así. Un ejercicio interesante es comprobar qué ocurre cuando se introduce un carácter en mayúscula.

5. Escribir un programa que lea los valores de los catetos de un triángulo rectángulo y calcule cuál es la hipotenusa, el área y el perímetro del triángulo mediante las siguientes expresiones:

$$h = \sqrt{c_1^2 + c_2^2} \qquad A = \frac{c_1 * c_2}{2} \qquad p = h + c_1 + c_2$$

Análisis

Este problema consiste en obtener distintos cálculos de un triángulo rectángulo a partir de sus catetos. Es decir, en este caso, hay dos datos de entrada (cada uno de los catetos) y tres datos de salida (hipotenusa, área y perímetro). El proceso para obtener los datos de salida a partir de los datos de entrada se resume en el enunciado en forma de expresiones a aplicar. Por tanto, en el primer nivel de análisis se determina:

1. datos de entrada: los dos catetos del triángulo rectángulo
2. datos de salida: hipotenusa, área y perímetro de este triángulo
3. proceso: obtener la hipotenusa, el área y el perímetro utilizando las expresiones proporcionadas en el enunciado.

En cuanto a la parte del proceso, es importante darse cuenta de que para calcular la hipotenusa y el área únicamente se utilizan los datos de entrada, mientras que en el cálculo del perímetro se necesita también uno de los datos de salida. Esto determina un orden entre las operaciones, es decir, se deberá calcular antes la hipotenusa que el perímetro.

De este modo, el programa deberá:

Leer los valores de los catetos del triángulo rectángulo y almacenarlos en las variables `cateto1` y `cateto2`.

Calcular la hipotenusa y el área y almacenar los resultados en las variables `hipotenusa` y `area` respectivamente.

Calcular el perímetro utilizando el contenido de las variables `cateto1`, `cateto2` e `hipotenusa` y almacenar el resultado en la variable `perimetro`.

Escribir por pantalla el contenido de las variables `hipotenusa`, `area` y `perimetro`.

En este caso, todas las variables pueden contener valores de tipo real, por lo que todas serán de tipo `float`.

Una cuestión importante es determinar cómo se pueden escribir las expresiones que aparecen en el enunciado en C. Tanto en la expresión para calcular el área como en la expresión para calcular el perímetro solamente aparecen operaciones como la suma, el producto y la división, que ya se han implementado en problemas anteriores. Pero para resolver la expresión $h = \sqrt{c_1^2 + c_2^2}$ es necesario el cálculo del cuadrado y de la raíz cuadrada de un número.

El cuadrado de un número `x` es sencillo de implementar: basta con multiplicar `x` por sí mismo, es decir, `cuadrado_x = x*x;` Sin embargo, para calcular la raíz cuadrada de un número `x` es necesario utilizar una instrucción especial que ofrece el lenguaje C: `sqrt`. Por tanto, la instrucción que calculará la raíz cuadrada de `x` es: `raiz_cuadrada_x = sqrt(x);`. Además, para poder utilizar esta instrucción especial se debe añadir al principio del programa (después de la línea `#include <stdio.h>`) la siguiente línea: `#include <math.h>`. Esto permite utilizar en el programa un conjunto de operaciones matemáticas como seno, coseno, potencia, etc.

En este caso, la expresión a escribir en C es: $h = \sqrt{c_1^2 + c_2^2}$. Si se escribe paso a paso, se obtiene:

```
cuadrado_cateto_1 = cateto1*cateto1;
cuadrado_cateto_2 = cateto2*cateto2;
suma_cuadrados_catetos = cuadrado_cateto_1+cuadrado_cateto_2;
hipotenusa = sqrt(suma_cuadrados_catetos);
```

Sin embargo, es posible simplificar este segmento de código eliminando la variable `suma_cuadrados_catetos`:

```
cuadrado_cateto_1 = cateto1*cateto1;
cuadrado_cateto_2 = cateto2*cateto2;
hipotenusa = sqrt(cuadrado_cateto_1+cuadrado_cateto_2;);
```

Todavía es posible simplificarlo más, calculando el cuadrado de los catetos directamente al obtener la hipotenusa:

```
hipotenusa=sqrt(cateto1*cateto1 + cateto2*cateto2);
```

Implementación

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

/* Programa que calcula la hipotenusa, el area y el perimetro
de un triangulo rectangulo a partir de sus catetos */

main()
{
float cateto1, cateto2, hipotenusa, area, perimetro;

/* Lectura de datos de entrada: cateto1 y cateto2 */
printf ("Cateto1? ");
scanf ("%f", &cateto1);
printf ("Cateto2? ");
scanf ("%f", &cateto2);
```

```
/* Proceso: calculo de la hipotenusa, del area y del
perimetro */
hipotenusa=sqrt(cateto1*cateto1 + cateto2*cateto2);

area=(cateto1*cateto2)/2.0;

perimetro=cateto1+cateto2+hipotenusa;

/* Salida de los resultados */
printf ("La hipotenusa es: %f\n", hipotenusa);
printf ("El area es: %f\n", area);
printf ("El perimetro es: %f\n", perimetro);
}
```

6. La empresa que fabrica un modelo de máquinas expendedoras de refrescos necesita un programa para estas máquinas que realice el cálculo de cuántas monedas de cada tipo debe devolver. Para ello, en primer lugar, se introducirá por teclado la cantidad a devolver en euros (múltiplo de 5 céntimos, que es la moneda más pequeña de la que se dispone), es decir, se tecleará 1.85 para 1 euro con 85 céntimos. Este programa escribirá en pantalla cuántas monedas de cada tipo hay que devolver teniendo en cuenta que:

   - Se dispone de monedas de 50 céntimos, 20 céntimos, 10 céntimos y 5 céntimos.
   - Siempre se dispone de las monedas necesarias de cada tipo.
   - Se debe devolver el menor número de monedas posible, es decir, intentar devolver con las de mayor valor.

   Ejemplos:

   - Si se introduce la cantidad de 1 euro con 85 céntimos, el programa debe imprimir: 3 monedas de 50 céntimos, 1 moneda de 20 céntimos, 1 moneda de 10 céntimos, 1 moneda de 5 céntimos.
   - Si se introduce la cantidad de 1 euro con 20 céntimos, el programa debe imprimir: 2 monedas de 50 céntimos, 1 moneda de 20 céntimos, 0 monedas de 10 céntimos, 0 monedas de 5 céntimos.

Análisis

Este problema consiste en determinar cuántas monedas de cada tipo se necesitan para formar la cantidad que debe devolver la máquina expendedora, teniendo en cuenta que se debe utilizar el menor número posible de monedas de cada tipo. Esto supone que, hay un único dato de entrada (la cantidad a devolver), mientras que se tienen varios datos de salida (el número de monedas de cada tipo). Así:

- datos de entrada: cantidad a devolver
- datos de salida: cuántas monedas de 50 céntimos, cuántas de 20 céntimos, cuántas de 10 céntimos y cuántas de 5 céntimos se necesitan
- proceso: determinar cuántas monedas de cada tipo hacen falta para formar la cantidad a devolver.

Una vez determinados los datos de entrada y de salida, habrá que pensar cómo es posible calcular el número de monedas de cada tipo que hacen falta. Un dato muy importante es el hecho de que se desea utilizar el menor número de monedas posible, lo que indica que se debe comenzar a emplear las monedas de mayor valor. Así, supongamos que la cantidad a devolver es 1.85 euros. Si se divide esta cantidad por 0.5 euros (que es el valor de la moneda de 50 céntimos) se obtiene 3.7, lo que significa que se necesitan 3 monedas de 50 céntimos y todavía queda cantidad por devolver. Es decir, ya se ha devuelto 1.50 euros con 3 monedas de 50 céntimos y quedan 0.35 euros (1.85 – 3*0.50) por devolver. Ahora si se divide 0.35 euros entre 0.2 euros (la moneda de 20 céntimos), se obtiene 1.75, por lo que se utiliza una moneda de 20 céntimos y todavía quedan 0.15 euros (0.35 – 1*0.20) por devolver. Si se divide 0.15 euros entre 0.1 euros (la moneda de 10 céntimos), el resultado es 1.5, lo que significa que se utiliza una moneda de 10 céntimos y quedan 0.05 euros por devolver, que al dividirlo por 0.05 euros (la moneda de 5 céntimos), se obtiene 1, es decir, una moneda de 5 céntimos y ya no queda nada por devolver.

Este proceso puede resumirse de la siguiente forma:

- dividir la cantidad a devolver por 0.5 para obtener el número de monedas de 50 céntimos a utilizar
- calcular lo que queda por devolver restando la cantidad a devolver menos el número de monedas de 50 céntimos utilizadas por 0.5 (que es su valor en euros)
- dividir la cantidad que queda por devolver por 0.2 para obtener el número de monedas de 20 céntimos a utilizar
- calcular lo que queda por devolver restando lo que quedaba por devolver menos el número de monedas de 20 céntimos utilizadas por 0.2 (que es su valor en euros)
- dividir la cantidad que queda por devolver por 0.1 para obtener el número de monedas de 10 céntimos a utilizar

- calcular lo que queda por devolver restando lo que quedaba por devolver menos el número de monedas de 10 céntimos utilizadas por 0.1 (que es su valor en euros)
- dividir la cantidad que queda por devolver por 0.05 para obtener el número de monedas de 5 céntimos a utilizar.

Por tanto, el programa deberá:

1. Leer la cantidad a devolver en euros y almacenarlo en la variable `euros_a_devolver`. Copiar el contenido de `euros_a_devolver` en la variable `falta_por_devolver`, ya que todavía queda todo por devolver.
2. Dividir la variable `falta_por_devolver` entre 0.5 y almacenar la parte entera de la división en la variable `monedas_50c`.
3. Obtener la cantidad que queda por devolver restando la variable `falta_por_devolver` menos la variable `monedas_50c` por 0.5. Almacenar el resultado en la variable `falta_por_devolver`.
4. Dividir la variable `falta_por_devolver` entre 0.2 y almacenar la parte entera de la división en la variable `monedas_20c`.
5. Obtener la cantidad que queda por devolver restando la variable `falta_por_devolver` menos la variable `monedas_20c` por 0.2. Almacenar el resultado en la variable `falta_por_devolver`.
6. Dividir la variable `falta_por_devolver` entre 0.1 y almacenar la parte entera de la división en la variable `monedas_10c`.
7. Obtener la cantidad que queda por devolver restando la variable `falta_por_devolver` menos la variable `monedas_10c` por 0.1. Almacenar el resultado en la variable `falta_por_devolver`.
8. Dividir la variable `falta_por_devolver` entre 0.05 y almacenar la parte entera de la división en la variable `monedas_5c`.
9. Escribir en pantalla el contenido de las variables `monedas_50c`, `monedas_20c`, `monedas_10c` y `monedas_5c`.

Es el momento de determinar el tipo de las variables anteriores:

- `euros_a_devolver` y `falta_por_devolver` contendrán la cantidad en euros que queda por devolver inicialmente y después de cada cálculo, que son datos de tipo real y por ello se deben declarar de tipo `float`.
- las variables `monedas_50c`, `monedas_20c`, `monedas_10c` y `monedas_5c` almacenan el número de monedas de cada tipo que se van a utilizar, es decir, datos de tipo entero, por lo que se deben declarar de tipo `int` (no superarán el rango máximo de este tipo).

Implementación

```
#include <stdio.h>

/*Programa que calcula el cambio en una maquina expendedora*/

main()
{
float euros_a_devolver, falta_por_devolver;
int monedas_50c, monedas_20c, monedas_10c, monedas_5c;

/* Lectura de datos: euros_a_devolver */

printf ("Euros a devolver: ");
scanf ("%f", &euros_a_devolver);

/* Proceso */

falta_por_devolver=euros_a_devolver;

monedas_50c=falta_por_devolver/0.5;
falta_por_devolver=falta_por_devolver-0.5*monedas_50c;

monedas_20c=falta_por_devolver/0.2;
falta_por_devolver=falta_por_devolver-0.2*monedas_20c;

monedas_10c=falta_por_devolver/0.1;
falta_por_devolver=falta_por_devolver-0.1*monedas_10c;

monedas_5c=falta_por_devolver/0.05;

/* Salida */

printf ("La cantidad de %f euros se devolvera asi:\n",
euros_a_devolver);
printf ("- %d monedas de 50 centimos\n", monedas_50c);
printf ("- %d monedas de 20 centimos\n", monedas_20c);
printf ("- %d monedas de 10 centimos\n", monedas_10c);
printf ("- %d monedas de 5 centimos\n", monedas_5c);
}
```

Es importante destacar el hecho de que cuando se realiza el cálculo del número de monedas de 50 céntimos con la instrucción:

```
monedas_50c=falta_por_devolver/0.5;
```

la división se realiza entre dos valores reales, luego el valor obtenido será un valor real, pero al ser almacenado en una variable entera, automáticamente se elimina la parte decimal.

El siguiente programa resuelve el mismo problema, pero se diferencia del programa anterior en que los cálculos se realizan en céntimos en lugar de en euros. Esto implica que la variable `falta_por_devolver` es ahora de tipo `int`. Además, las divisiones para calcular el número de monedas de cada tipo, son ahora divisiones enteras ya que ambos operandos lo son:

```
monedas_50c=falta_por_devolver/50;
```

```
#include <stdio.h>

/*Programa que calcula el cambio en una maquina expendedora*/

main()
{
float euros_a_devolver;
int monedas_50c, monedas_20c, monedas_10c, monedas_5c;
int falta_por_devolver;

/* Lectura de datos: euros_a_devolver */

printf ("Euros a devolver: ");
scanf ("%f", &euros_a_devolver);

/* Proceso */
falta_por_devolver=euros_a_devolver*100.0;

monedas_50c=falta_por_devolver/50;
falta_por_devolver=falta_por_devolver-50*monedas_50c;

monedas_20c=falta_por_devolver/20;
falta_por_devolver=falta_por_devolver-20*monedas_20c;
```

```
monedas_10c=falta_por_devolver/10;
falta_por_devolver=falta_por_devolver-10*monedas_10c;

monedas_5c=falta_por_devolver/5;

/* Salida */
printf ("La cantidad de %f euros se devolvera asi:\n",
euros_a_devolver);
printf ("- %d monedas de 50 centimos\n", monedas_50c);
printf ("- %d monedas de 20 centimos\n", monedas_20c);
printf ("- %d monedas de 10 centimos\n", monedas_10c);
printf ("- %d monedas de 5 centimos\n", monedas_5c);
}
```

3. Problemas Resueltos

7. Indicar cuál será el resultado de la ejecución de este programa:

```
#include <stdio.h>
main()
{
int x=3, y;
float z;
y=x+3;
printf (“%d\n”, y);
y=3*x*x + 2*x + 1;
printf (“%d\n”, y);
z=x*0.3;
printf (“%f\n”, z);
z=y/x;
printf (“%f\n”, z);
z=(float)y/x;
printf (“%f\n”, z);
}
```

Solución

Este programa mostrará en pantalla los siguientes resultados:

```
6
34
0.900000
11.000000
11.333333
```

8. Una compañía de refrescos comercializa tres productos: de cola, de naranja y de limón. Se desea realizar un programa que calcule las ventas realizadas de cada producto. Para ello, se leerá la cantidad vendida (máximo 5000000) y el precio en euros de cada producto y se mostrará un informe de ventas como el que sigue:

```
Producto      Ventas      Precio          Total
-------------------------------------------
Cola         1000000        0.17      170000.00
Naranja       350000        0.20       70000.00
Limon         530000        0.19      100700.00
                           TOTAL      340700.00
```

Para resolver este problema es necesario leer las ventas y el precio de cada uno de los productos y multiplicar los valores leídos para obtener el total. La salida debe mostrarse tabulada, utilizando especificaciones de formato más elaboradas en las instrucciones `printf`. El programa que resuelve este problema es el siguiente:

```
#include <stdio.h>

/* Programa que muestra el total de ventas de varios
productos de una empresa de refrescos */

main()
{
long int ventas_cola, ventas_naranja, ventas_limon;
float precio_cola, precio_naranja, precio_limon;
float total_cola, total_naranja, total_limon;
```

```
/* Lectura de datos: ventas y precio de cada producto */
printf ("Datos de la cola:\n");
printf ("\tVentas:");
scanf ("%ld", &ventas_cola);
printf ("\tPrecio: ");
scanf ("%f", &precio_cola);

printf ("Datos de la naranja:\n");
printf ("\tVentas:");
scanf ("%ld", &ventas_naranja);
printf ("\tPrecio: ");
scanf ("%f", &precio_naranja);

printf ("Datos del limon:\n");
printf ("\tVentas:");
scanf ("%ld", &ventas_limon);
printf ("\tPrecio: ");
scanf ("%f", &precio_limon);

/* Proceso */

total_cola = ventas_cola*precio_cola;
total_naranja = ventas_naranja*precio_naranja;
total_limon = ventas_limon*precio_limon;

/* Salida tabulada */

printf ("Producto      Ventas      Precio         Total\n");
printf ("-----------------------------------------------\n");
printf ("Cola         %7ld        %4.2f     %9.2f\n",
ventas_cola, precio_cola, total_cola);
printf ("Naranja      %7ld        %4.2f     %9.2f\n",
ventas_naranja, precio_naranja, total_naranja);
printf ("Limon        %7ld        %4.2f     %9.2f\n",
ventas_limon, precio_limon, total_limon);
printf ("                           TOTAL     %9.2f\n",
 total_cola+total_naranja+total_limon);
}
```

9. Escribir un programa que muestra el resultado de la ecuación de tercer grado $y = ax^3 + bx^2 + cx + d$ para un valor de x. Para ello, debe leer el valor de los coeficientes (*a, b, c y d*) y el valor de *x* y mostrar por pantalla el resultado de la evaluación de la ecuación resultante.

El programa que resuelve este problema debe, en primer lugar, leer cada uno de los coeficientes del polinomio y el valor de x. A continuación, debe calcular el valor final del polinomio escribiendo una expresión que lo represente. Por último, debe mostrar el valor resultante. La implementación de este programa es la siguiente:

```
#include <stdio.h>

/* Programa que muestra el valor de y=ax^3 + bx^2 + cx + d */

main()
{
float a, b, c, d, x, y;

/* Lectura de coeficientes */
printf("Evaluacion del polinomio y=ax^3 + bx^2 + cx + d.\n");
printf("Valor de los coeficientes a, b, c y d:");
scanf ("%f%f%f%f", &a, &b, &c, &d);
/* Lectura del valor de x */
printf ("Introducir el valor de la variable x:");
scanf ("%f", &x);

/* Proceso: calculo del polinomio */
y = a*x*x*x + b*x*x + c*x + d;

/* Salida */
printf ("El resultado del polinomio %.2f x^3 + %.2f x^2", a,
b);
printf (" + %.2f x + %.2f para x=%.2f es: %.2f\n", c, d, x,
y);
}
```

4. Problemas Propuestos

10. Indicar cuál será el resultado de la ejecución del siguiente programa:

```
main() {
int x=7, y;
y=-2+x;
printf (“%d\n”, y);
y=y+2;
printf (“%d\n”, y);
y=(y==x);
printf (“%d\n”, y);
y++;
printf (“%d\n”, y);
}
```

11. Evaluar las siguientes expresiones:

```
a) 5 / 2 + 20 % 6
b) 4 * 6 / 2 - 15 / 2
c) 5 * 15 / 2 / (4 - 2)
d) 8 == 16 || 7 != 4 && 4 < 1
e) (4 * 3 < 6 || 3 > 5 - 2) && 3 + 2 < 12
f) 7 + 3 * 6 / 2 - 1
g) 2 % 2 + 2 * 2 - 2 / 2
h) (3 * 9 * (3 + (9 * 3 / (3))))
```

12. Evaluar las siguientes expresiones lógicas y determinar su valor final, suponiendo que las variables a, b, c, x, y, z contienen los valores 4, 3, 4, 1, 1, 1, respectivamente.

```
a) h = (a > b) || (b == c) ;
b) h = (a != b) && (a != b) || (x == z);
c) h = ((a >= b) || (x >= z)) && ((x != b) && (y == z));
d) h = (!(a > b)) || (x != b);
```

13. Escribir un programa que permita resolver una ecuación de primer grado $ax+b=c$ introduciendo los coeficientes *a*, *b* y *c* por teclado.

14. Modificar el programa anterior para que el resultado final sea:
    - a toma el valor inicial de b
    - b toma el valor inicial de c
    - c toma el valor inicial de a.

    Es decir, si se introducen los valores 1, 2 y 3 para a, b y c respectivamente, la salida por pantalla será:

    ```
    El resultado es: 2  3  1
    ```

15. ¿Qué resultado aparecerá en pantalla al ejecutar el siguiente programa, si se introducen los valores 1, 2 y 3 para a, b y c respectivamente?

```
#include <stdio.h>
main() {
int a, b, c;
printf ("Introduce tres números: ");
scanf ("%d%d%d", &a, &b, &c);
a=b; b=c; c=a;
printf ("El resultado es: %d  %d  %d\n", a, b, c);
}
```

16. Escribir un programa que calcule el área y el perímetro de un rectángulo y muestre el resultado en pantalla, sabiendo que:

    $$area = base * altura \qquad perimetro = 2 * base + 2 * altura$$

17. Escribir un programa que lea el número que ha salido en el sorteo de la ONCE e imprima en pantalla la última cifra de este número.

18. Escribir un programa que simule a una calculadora sencilla. Este programa pedirá dos números por teclado y calculará la suma, la resta, el producto y la división de ambos.

19. Escribir un programa que calcule el área y el perímetro de una circunferencia de radio R introducido por el usuario. Las fórmulas a aplicar son:

$$area = \Pi R^2$$
$$perimetro = 2\Pi R$$

20. Escribir un programa que lea los valores de tres resistencias eléctricas (en Ohmios, W) y muestre en pantalla el valor global de la resistencia formada por estas tres resistencias si:

    a) están conectadas en paralelo: $R = \dfrac{1}{1/R_1 + 1/R_2 + 1/R_3}$

    b) están conectadas en serie: $R = R_1 + R_2 + R_3$

21. El servicio de endocrinología de un hospital necesita un programa para calcular el peso recomendado de una persona. Escribir un programa que lea la altura en metros y la edad de una persona y realice el cálculo del peso recomendado según la siguiente fórmula:

    peso = (altura en centímetros – 100 + 10% de la edad) * 0.9

22. Un asesor nos ha solicitado un programa para calcular los pagos mensuales de una hipoteca, de manera que pueda asesorar a sus clientes sobre ello. El programa debe solicitar el capital del préstamo (C), el interés anual (I) y el número de años (N) de la hipoteca y debe escribir la cuota a pagar mensualmente. Para calcular esta cuota se utiliza la siguiente fórmula, donde R es el interés mensual:

$$cuota = \frac{C * R}{1 - \left(\dfrac{1}{1+R}\right)^{N.12}} \qquad R = \frac{I/100}{12}$$

23. Escribir un programa que calcule un determinante de 2º orden, sabiendo que:

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = a * d - c * b$$

24. Escribir un programa que resuelva sistemas de ecuaciones del siguiente tipo:

$$\left.\begin{aligned} ax + by = c \\ a'x + b'y = c' \end{aligned}\right\}$$

Este tipo de sistemas puede resolverse aplicando las siguientes expresiones:

$$y = \frac{a'c - ac'}{a'b - ab'} \qquad x = \frac{c - by}{a}$$

25. En un Campeonato de Atletismo se miden las marcas de los corredores de las distintas categorías en segundos porque resultan más apropiados para almacenarlos en un soporte informático. Sin embargo, resulta muy incómodo para los aficionados leer las marcas en segundos. Por esta razón, se desea un programa que lea un tiempo expresado en segundos y muestre su equivalencia en horas, minutos y segundos. Por ejemplo, si se introduce 4550 segundos, debe mostrar: 1 hora, 15 minutos y 50 segundos.

26. Escribir un programa que obtiene el valor de *y* de la siguiente ecuación, para un valor de *x* introducido por teclado:

$$y = \frac{x^2 - 4}{2} + \frac{3x - 7x^4}{-5x^3} + 4x - 2$$

27. La franquicia de tiendas de ropa Raza nos ha pedido que realicemos un programa para su terminal punto de venta. El programa deberá solicitar el código del artículo (máximo 6 cifras) a vender, su precio en euros, la cantidad de artículos que se

Para seguir leyendo, inicie el proceso de compra, click aquí