CAPÍTULO 7

CARACTERES Y CADENAS DE CARACTERES

Ya hemos visto que un carácter se codifica en C mediante el tipo de dato ***char***. Es una posición de memoria (la más pequeña que se puede reservar en C: un byte) que codifica cada carácter según un código arbitrario. El más extendido en el mundo del PC y en programación con lenguaje C es el código ASCII.

Hasta el momento, cuando hemos hablado de los operadores de una variable ***char***, hemos hecho referencia al uso de esa variable (que no se ha recomendado) como entero de pequeño rango o dominio. Pero donde realmente tiene uso este tipo de dato es en el manejo de caracteres y en el de vectores o arrays declarados de este tipo.

A un array de tipo ***char*** se le suele llamar **cadena de caracteres**.

En este capítulo vamos a ver las operaciones básicas que se pueden realizar con caracteres y con cadenas. No hablamos de operadores,

porque estos ya se han visto antes, en un capítulo previo, y se reducen a los aritméticos, lógicos, relacionales y a nivel de bit. Hablamos de operaciones habituales cuando se manejan caracteres y sus cadenas: operaciones que están definidas como funciones en algunas bibliotecas del ANSI C y que presentaremos en este capítulo.

Operaciones con caracteres.

La biblioteca **ctype.h** contiene abundantes funciones para la manipulación de caracteres. En la ayuda on line que ofrece cualquier editor de C se pueden encontrar indicaciones concretas y prácticas para el uso de cada una de ellas.

La biblioteca **ctype.h** define funciones de manejo de caracteres de forma individual, no concatenados formando cadenas.

Lo más indicado será ir viendo cada una de esas funciones y explicar qué operación realiza sobre el carácter y qué valores devuelve.

- ***int isalnum(int c);***

  Recibe el código ASCII de una carácter y devuelve el valor 1 si el carácter que corresponde a ese código ASCII es una letra o un dígito numérico; en caso contrario devuelve un 0.

  Ejemplo de uso:

```
if(isalnum('@')) printf("Alfanumérico"):
else printf("No alfanumérico");
```

  Así podemos saber si el carácter '@' es considerado alfabético o numérico. La respuesta será que no lo es. Evidentemente, para hacer uso de esta función y de todas las que se van a ver en este epígrafe, hay que indicar al compilador la biblioteca en donde se encuentran definidas estas bibliotecas: ***#define <ctype.h>***.

- ***int isalpha(int c);***

Recibe el código ASCII de una carácter y devuelve el valor 1 si el carácter que corresponde a ese código ASCII es una letra; en caso contrario devuelve un 0.

Ejemplo de uso:

```
if(isalnum('2')) printf("Alfabético"):
else printf("No alfabético");
```

La respuesta será que '2' no es alfabético.

- ***int iscntrl(int c);***

  Recibe el código ASCII de una carácter y devuelve el valor 1 si el carácter que corresponde a ese código ASCII es el carácter borrado o un carácter de control (ASCII entre 0 y 1F, y el 7F, en hexadecimal); en caso contrario devuelve un 0.

  Ejemplo de uso:

```
if(iscntrl('\n')) printf("Carácter de control"):
else printf("No carácter de control");
```

  La respuesta será que el salto de línea sí es carácter de control.

  Resumidamente ya, presentamos el resto de funciones de esta biblioteca. Todas ellas reciben como parámetro el código ASCII de un carácter.

- ***int isdigit(int c);*** Devuelve el valor 1 si el carácter es un dígito; en caso contrario devuelve un 0.
- ***int isgraph(int c);*** Devuelve el valor 1 si el carácter es un carácter imprimible; en caso contrario devuelve un 0.
- ***int isascii(int c);*** Devuelve el valor 1 si el código ASCII del carácter es menor de 128; en caso contrario devuelve un 0.
- ***int islower(int c);*** Devuelve el valor 1 si el carácter es una letra minúscula; en caso contrario devuelve un 0.
- ***int ispunct(int c);*** Devuelve el valor 1 si el carácter es signo de puntuación; en caso contrario devuelve un 0.

- ***int isspace(int c);*** Devuelve el valor 1 si el carácter es el espacio en blanco, tabulador vertical u horizontal, retorno de carro, salto de línea, u otro carácter de significado espacial en un texto; en caso contrario devuelve un 0.
- ***int isupper(int c);*** Devuelve el valor 1 si el carácter es una letra mayúscula; en caso contrario devuelve un 0.
- ***int isxdigit(int c);*** Devuelve el valor 1 si el carácter es un dígito hexadecimal (del '0' al '9', de la 'a' a la 'f' ó de la 'A' a la 'F'); en caso contrario devuelve un 0.
- ***int tolower(int ch);*** Si el carácter recibido es una letra mayúscula, devuelve el ASCII de su correspondiente minúscula; en caso contrario devuelve el mismo código ASCII que recibió como entrada.
- ***int toupper(int ch);*** Si el carácter recibido es una letra minúscula, devuelve el ASCII de su correspondiente mayúscula; en caso contrario devuelve el mismo código ASCII que recibió como entrada.

Con estas funciones definidas se pueden trabajar muy bien muchas operaciones que se pueden realizar con caracteres. Por ejemplo, veamos un programa que solicita caracteres por consola, hasta que recibe el carácter salto de línea, y que únicamente muestra por pantalla los caracteres introducidos que sean letras. Al final indicará cuántas veces han sido pulsadas cada una de las cinco vocales.

```
#include <stdio.h>
#include <ctype.h>
void main(void)
{
	unsigned short int a = 0, e = 0, i = 0, o = 0, u = 0;
	char ch;
	do
	{
		ch = getchar();
		if(isalpha(ch))
		{
			if(ch == 'a') a++;
			else if(ch == 'e') e++;
			else if(ch == 'i') i++;
			else if(ch == 'o') o++;
```

```
                else if(ch == 'u') u++;
            }
            else
                printf("\b ");
        }while(ch != 10);
        printf("\n\nVocal a ... %hu",a);
        printf("\nVocal e ... %hu",e);
        printf("\nVocal i ... %hu",i);
        printf("\nVocal o ... %hu",o);
        printf("\nVocal u ... %hu",u);
}
```

Si el carácter introducido es alfabético, entonces simplemente verifica si es una vocal y aumenta el contador particular para cada vocal. Si no lo es, entonces imprime en pantalla un retorno de carro y un carácter blanco, de forma que borra el carácter que había sido introducido mediante la función *getchar* y que no deseamos que salga en pantalla.

El carácter intro es el ASCII 13. Así lo hemos señalado en la condición que regula la estructura ***do–while***.

Entrada de caracteres.

Hemos visto dos funciones que sirven bien para la introducción de caracteres.

La función *scanf* espera la entrada de un carácter más el carácter intro. No es muy cómoda cuando se espera únicamente un carácter.

La función *getchar* también está definida en la biblioteca **stdio.h**. De todas formas, su uso no es siempre el esperado, por problemas del buffer de teclado. Un buffer es como una cola o almacén que crea y gestiona el sistema operativo. Nuestro programas nunca actúan directamente sobre la máquina, sino siempre a través de los servicios que ofrece ese sistema operativo. Con frecuencia ocurre que la función *getchar* debería esperar la entrada por teclado de un carácter, pero no lo hace porque ya hay caracteres a la espera en el buffer gestionado por el sistema operativo.

Si se está trabajando con un editor en el sistema operativo Windows, se puede hacer uso de la biblioteca **conio.h** y de algunas de sus funciones de entrada por consola. Esta biblioteca no es estándar en ANSI C, pero si vamos a trabajar en ese sistema operativo sí resulta válida.

En esa biblioteca vienen definidas dos funciones útiles para la entrada por teclado, y que no dan los problemas que da, en Windows, la función *getchar*.

Esas dos funciones son:

int getche(void); espera del usuario un pulso de teclado. Devuelve el código ASCII del carácter pulsado y muestra por pantalla ese carácter. Al ser invocada esta función, no recibe valor o parámetro alguno: por eso se define como de tipo ***void***.

int getch(void); espera del usuario un pulso de teclado. Devuelve el código ASCII del carácter pulsado. Al ser invocada esta función, no recibe valor o parámetro alguno: por eso se define como de tipo ***void***. Esta función no tiene eco en pantalla, y no se ve el carácter pulsado.

Cadena de caracteres.

Una cadena de caracteres es una formación de caracteres. Es un vector tipo ***char***, cuyo último elemento es el carácter nulo (ó NULL, ó '\0' se escribe en C). Toda cadena de caracteres termina con el carácter llamado carácter NULO de C. Este carácter indica donde termia la cadena.

La declaración de una cadena de caracteres se realiza de forma similar a la de un vector de cualquier otro tipo:

char mi_cadena[dimensión];

donde dimensión es un literal que indica el número de bytes que se deben reservar para la cadena (recuerde que una variable tipo ***char*** ocupa un byte).

La asignación de valores, cuando se crea una cadena, puede ser del mismo modo que la asignación de vectores:

```
char mi_cadena[4] = {'h','o','l','a'};
```

Y así hemos asignado a la variable *mi_cadena* el valor de la cadena "hola".

Y así, con esta operación de asignación, acabamos de cometer un error importante. Repasemos… ¿qué es una cadena?: es un vector tipo ***char***, cuyo último elemento es el carácter nulo. Pero si el último elemento es el carácter nulo… ¿no nos hemos equivocado en algo?: Sí.

La correcta declaración de esta cadena sería:

```
char mi_cadena[5] = {'h','o','l','a','\0'};
```

Faltaba el carácter nulo con el que debe terminar toda cadena. De todas formas, la asignación de valor a una cadena suele hacerse mediante **comillas dobles**, de la siguiente manera:

```
char mi_cadena[5] = "hola";
```

Y ya en la cadena "*hola*" se recoge el quinto carácter: el carácter nulo. Ya se encarga el compilador de C de introducir el carácter nulo al final de la cadena.

Y es que hay que distinguir, por ejemplo, entre el carácter 'a' y la cadena "a". En el primer caso tratamos del valor ASCII 97, que codifica la letra 'a' minúscula; en el segundo caso tratamos de una cadena de dos caracteres: el carácter 'a' seguido del carácter nulo.

También podríamos haber hecho lo siguiente:

```
char mi_cadena[100] = "hola";
```

donde todos los bytes posteriores al carácter nulo tendrán valores aleatorios, no asignados. Pero eso no importa, porque la cadena sólo se extiende hasta el carácter nulo: no nos interesa lo que pueda haber más allá de ese carácter.

Y al igual que con los arrays, podemos inicializar la cadena, sin necesidad de dimensionarla:

```
char mi_cadena[] = "hola";
```

Y entonces ya se encarga el compilador de reservar cinco bytes para la variable *mi_cadena*.

Una forma de vaciar una cadena y asignarle el valor de cadena vacía es el siguiente;

```
mi_cadena[0] = '\0';
```

donde, ya lo hemos dicho, '\0' es el carácter nulo.

Definimos **longitud de la cadena** como el número de caracteres previos al carácter nulo. El carácter nulo no cuenta como parte para el cálculo de la longitud de la cadena. La cadena "hola" necesita cinco variables ***char*** para ser almacenada, pero su longitud se dice que es cuatro. Asignando al elemento de índice 0 el valor nulo, tenemos una cadena de longitud cero.

Cada elemento de la cadena se reconoce a través del índice entre corchetes. Cuando se quiere hacer referencia a toda la cadena en su conjunto, se emplea el nombre sin ningún corchete ni índice.

Dar valor a una cadena de caracteres.

Para recibir cadenas por teclado disponemos, entre otras, de la función *scanf* ya presentada en capítulos anteriores. Esa función toma la cadena introducida por teclado, pero la corta a partir de la primera entrada de un carácter en blanco. Se puede evitar ese corte, pero en general, para introducir por teclado una cadena que puede tener caracteres en blanco, muchos compiladores de C recomiendan el uso de otra función, mucho más cómoda, que es la función ***gets***.

La función *gets* está definida en la biblioteca **stdio.h**, y su prototipo es el siguiente:

char *gets(char *s);

Esta función asigna a la cadena *s* todos los caracteres introducidos como cadena. La función queda a la espera de que el usuario introduzca la cadena de texto. Hasta que no se pulse la tecla intro, se supone que todo lo que se teclee será parte de la cadena de entrada. Al final de todos ellos, como es natural, coloca el carácter nulo.

Hay que hacer una advertencia grave sobre el uso de esta función: puede ocurrir que la cadena introducida por teclado sea de mayor longitud que el número de bytes que se han reservado. Esa incidencia no es vigilada por la función *gets*. Y si ocurre, entonces, además de grabar información de la cadena en los bytes reservados para ello, se hará uso de los bytes, inmediatamente consecutivos a los de la cadena, hasta almacenar toda la información tecleada más su carácter nulo final. Esa violación de memoria puede tener —y de hecho habitualmente tiene— consecuencias desastrosas para el programa.

Ejemplo: programa que pregunta el nombre y, entonces saluda al usuario.

```
#include <stdio.h>
void main(void)
{
      char nombre[10];
      printf("¿Cómo te llamas?  ");
      gets(nombre);
      printf("Hola, %s.", nombre);
}
```

El programa está bien construido. Pero hay que tener en cuenta que el nombre que se introduce puede, fácilmente, superar los 10 caracteres. Por ejemplo, si un usuario responde diciendo “José Antonio”, ya ha introducido 13 caracteres: 4 por José, 7 por Antonio, 1 por el carácter en blanco, y otro más por el carácter nulo final. En ese caso, el comportamiento del programa sería imprevisible.

Se puede hacer un programa que acepte la entrada por teclado de carácter a carácter, y vaya mostrando la entrada por pantalla a la vez que la va almacenando en la cadena. Veamos una posible implementación.

```
#define TAM 30
#include <stdio.h>
#include <ctype.h>
void main(void)
{
      char nombre[TAM];
      short int i;
      printf("¿Cómo te llamas?  ");
      for(i = 0 ; i < TAM ; i++)
            nombre[i] = NULL;
      i = 0;
      while(i < TAM)
      {
            nombre[i] = getchar();
            if (nombre[i] == 10)
            {
                  nombre[i] = NULL;
                  break;
            }
            else if(nombre[i] == 8 && i > 0)
            {
                  nombre[i] = NULL;
                  printf("\b \b");
                  i--;
            }
            else if(isgraph(nombre[i]) || nombre[i] == 32)
                  printf("%c",nombre[i++]);
      }
      printf("\n\nHola, %s.", nombre);
}
```

Donde el valor ASCII 8 es el carácter borrar uno hacia detrás: eso significa que hay que retroceder, escribir un blanco donde ya habíamos escrito otro carácter, y volver a retroceder; además hay que reducir en uno el valor del índice *i*, puesto que hemos eliminado un carácter. Si estábamos en el carácter 0, la operación de borrado no surte efecto, porque así lo hemos condicionado en el ***if*** correspondiente. El valor ASCII 10 es el carácter intro, que vamos a entender como final de entrada de la cadena: por eso se interrumpe el proceso de entrada de caracteres y se cambia el valor de ese último carácter que deja de ser

10 para ser el carácter nulo. Y el carácter 32 es el carácter espacio en blanco, que no se resuelve como imprimible en la función *isgraph*.

Esta pequeña aplicación, e incluso alguna un poco más desarrollada, podrían resolver el problema de la función *gets*, pero para nuestro estudio será mejor usar la función de **stdio.h** sin complicarse la vida.

Operaciones con cadenas de caracteres.

Todo lo visto en el capítulo de vectores es perfectamente aplicable a las cadenas: de hecho una cadena no es más que un vector de tipo ***char***. De todas formas, las cadenas de caracteres merecen un tratamiento diferente al presentado para el resto de vectores, ya que las operaciones que se pueden realizar con cadenas son muy diferentes a las que se realizan con vectores numéricos: concatenar cadenas, buscar una subcadena en una cadena dada, determinar cuál de dos cadenas es mayor alfabéticamente, etc. Vamos a ver algunos programas básicos de manejo de cadena, y posteriormente presentaremos un conjunto de funciones de biblioteca definidas para las cadenas, y que se encuentran en la biblioteca **string.h**.

- ***Copiar el contenido de una cadena en otra cadena:***

```
#include <stdio.h>
void main(void)
{
      char original[100];
      char copia[100];
      short int i = 0;
      printf(“Cadena original ... ”);
      gets(original);
      while(original[i] != NULL)
      {
            copia[i] = original[i];
            i++;
      }
      copia[i] = NULL;
      printf("Original: %s\n",original);
      printf("Copia: %s\n",copia);
}
```

Mientras no se llega al carácter nulo de *original*, se van copiando uno a uno los valores de las variables de la cadena en *copia*. Al final, cuando ya se ha llegado al nulo en original, habrá que cerrar también la cadena en *copia*, mediante un carácter nulo.

El carácter nulo se escribe NULL, ó '\0'. Ambas formas son equivalentes.

Esta operación también se puede hacer con una función de la biblioteca **string**: la función

char *strcpy(char *dest, const char *src);

que recibe como parámetros las dos cadenas, origen y destino, y devuelve la dirección de la cadena de destino. Con esta función, el programa antes presentado queda de la siguiente forma:

```
#include <stdio.h>
#include <string.h>
void main(void)
{
      char original[100];
      char copia[100];
      printf("Cadena original ... ");
      gets(original);
      strcpy(copia, original);
      printf("Original: %s\n",original);
      printf("Copìa: %s\n",copia);
}
```

- ***Determinar la longitud de una cadena:***

```
#include <stdio.h>
void main(void)
{
      char original[100];
      short int i = 0;
      printf("Cadena original ... ");
      gets(original);
      while(original[i] != NULL) i++;
      printf("%s tiene longitud %hd\n",original,i);
}
```

El contador *i* se va incrementando hasta llegar al carácter nulo. Así, en *i*, tenemos la longitud de la cadena.

Esta operación también se puede hacer con una función de la biblioteca **string**: la función

size_t strlen(const char *s);

que recibe como parámetro una cadena de caracteres y devuelve su longitud. Una variable tipo ***size_t*** es, para nosotros y ahora mismo, una variable de tipo entero.

```
#include <stdio.h>
#include <string.h>
void main(void)
{
      char original[100];
      short int i;
      printf(“Cadena original ... ”);
      gets(original);
      i = strlen(original);
      printf("%s tiene longitud %hd\n",original,i);
}
```

- ***Concatenar una cadena al final de otra:***

```
#include <stdio.h>
void main(void)
{
      char original[100];
      char concat[100];
      short int i = 0, j = 0;
      printf(“Cadena original ... ”);
      gets(original);
      printf(“Cadena a concatenar ... ”);
      gets(concat);
      while(original[i] != NULL) i++;
      while(concat[j] != NULL)
      {
            original[i] = concat[j];
            i++;
            j++;
      }
      original[i] = NULL;
      printf("Texto concatenado: %s\n",original);
}
```

Y de nuevo disponemos de una función en la biblioteca **string** que realiza la concatenación de cadenas:

char *strcat(char *dest, const char *src);

Que recibe como parámetros las cadenas destino de la concatenación y cadena fuente, y devuelve la dirección de la cadena que contiene la cadena original más la concatenada.

```
#include <stdio.h>
#include <string.h>
void main(void)
{
      char original[100];
      char concat[100];
      printf(“Cadena original ... ”);
      gets(original);
      printf(“Cadena a concatenar ... ”);
      gets(concat);
      strcat(original, concat);
      printf("Texto concatenado: %s\n",original);
}
```

También existe otra función, parecida a esta última, que concatena no toda la segunda cadena, sino hasta un máximo de caracteres, fijado por un tercer parámetro de la función:

char *strncat(char *dest, const char *src, size_t maxlen);

- ***Comparar dos cadenas e indicar cuál de ellas es mayor, o si son iguales:***

```
#include <stdio.h>
void main(void)
{
      char cad01[100];
      char cad02[100];
      short int i = 0;
      short int chivato = 0;
      printf(“Primera Cadena ... ”);
      gets(cad01);
      printf(“Segunda Cadena ... ”);
      gets(cad02);
      while(cad01[i] != NULL && cad02[i] != NULL)
      {
            if(cad01[i] > cad02[i])
            {
                  chivato = 1;
                  break;
            }
            else if(cad01[i] < cad02[i])
            {
                  chivato = 2;
```

```
                    break;
              }
              i++;
        }
        if(chivato == 1)
              printf("cadena01 > cadena02");
        else if(chivato == 2)
              printf("cadena02 > cadena01");
        else if(cad01[i] == NULL && cad02[i] != NULL)
              printf("cadena02 > cadena01");
        else if(cad01[i] != NULL && cad02[i] == NULL)
              printf("cadena01 > cadena02");
        else
              printf("cadenas iguales");
}
```

Y una vez más, disponemos de una función en la biblioteca **string** que realiza la comparación de cadenas:

int strcmp(const char *s1, const char *s2);

Que recibe como parámetros las cadenas a comparar y devuelve un valor negativo si s1 < s2; un valor positivo si s1 > s2; y un cero si ambas cadenas son iguales

```
#include <string.h>
#include <stdio.h>
void main(void)
{
      char c1[100] = "Texto de la cadena primera";
      char c2[100] = "Texto de la cadena segunda";
      short int comp;
      printf(“Primera Cadena ... ”);
      gets(c1);
      printf(“Segunda Cadena ... ”);
      gets(c2);
      comp = strcmp(c1,c2);
      if(comp < 0) printf("cadena02 > cadena01");
      else if(comp > 0) printf("cadena01 > cadena02");
      else printf("Cadenas iguales");
}
```

También existe una función que compara hasta una cantidad de caracteres señalado, es decir, una porción de la cadena:

int strncmp(const char *s1, const char *s2, size_t maxlen);

donde *maxlen* es el tercer parámetro, que indica hasta cuántos caracteres se han de comparar.

Podríamos seguir con otras muchas funciones de la biblioteca **string**. Hemos mostrado algunas de las operaciones con cadenas, con su función y sin ella, para presentar con ejemplos el modo habitual con el que se manejan las cadenas de caracteres. Hay mucha información sobre estas y otras funciones de la biblioteca **string** en cualquier ayuda on line de cualquier editor de C. Se recomienda consultar esa ayuda para obtener información sobre ellas.

Otras funciones de cadena.

Vamos a detenernos en la conversión de una cadena de caracteres, todos ellos numéricos, en la cantidad numérica, para poder luego operar con ellos. Las funciones que veremos en este epígrafe se encuentran definidas en otras bibliotecas: en la **stdlib.h** o en la biblioteca **math.h**.

- Convertir una cadena de caracteres (todos ellos dígitos o signo decimal) en un ***double***.

  double strtod(const char *s, char **endptr);

  Esta función convierte la cadena *s* en un valor ***double***. La cadena *s* debe ser una secuencia de caracteres que puedan ser interpretados como un valor double. La forma genérica en que se puede presentar esa cadena de caracteres es la siguiente:

  [ws] [sn] [ddd] [.] [ddd] [fmt[sn]ddd]

  donde [ws] es un espacio en blanco opcional; [sn] es el signo opcional (+ ó –); [ddd] son dígitos opcionales; [fmt] es el formato exponencial, también opcional, que se indica con las letras ‘e’ ó ‘E’; finalmente, el [.] es el carácter punto decimal, también opcional. Por ejemplo, valores válidos serían + 1231.1981 e-1; ó 502.85E2; ó + 2010.952.

La función abandona el proceso de lectura y conversión en cuanto llega a un carácter que no puede ser interpretable como un número real. En ese caso, se puede hacer uso del segundo parámetro para detectar el error que ha encontrado: aquí se recomienda que para el segundo parámetro de esta función se indique el valor nulo: en esta parte del libro aún no se tiene suficiente formación en C para poder comprender y emplear bien este segundo parámetro.

La función devuelve, si ha conseguido la transformación, el número ya convertido en formato ***double***.

Vamos a ver un ejemplo.

```
#include <stdio.h>
#include <stdlib.h>
void main(void)
{
      char entrada[80];
      double valor;
      printf("Número decimal ... ");
      gets(entrada);
      valor = strtod(entrada, NULL);
      printf("La cadena es %s ", entrada);
      printf("y el número es %lf\n", valor);
}
```

De forma semejante se comporta la función ***atof***, de la biblioteca **math.h**. Su prototipo es:

double atof(const char *s);

Donde el formato de la cadena de entrada es el mismo que hemos visto para la función *strtod*.

Consultando la ayuda del compilador se puede ver cómo se emplean las funciones ***strtol*** y ***strtoul***, de la biblioteca **stdlib.h**: la primera convierte una cadena de caracteres en un entero largo; la segunda es lo mismo pero el entero es siempre largo sin signo. Y también las funciones ***atoi*** y ***atol***, que convierte la cadena de caracteres a ***int*** y a ***long int*** respectivamente.

Como ejemplo de todo lo dicho, veamos un programa que acepta como entrada una cadena de la forma “x + y =” y muestra por pantalla el resultado de la operación. Desde luego, el desarrollo presentado es uno de los muchos posibles; y como se ha dicho ya en otras ocasiones en este manual, no necesariamente la mejor de las soluciones:

```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
void main(void)
{
      char e[80];
      char n[20];
      char op;
      short i = 0, j = 0;
      short est = 0, err = 0;
      long int a1, a2, res;
      printf("Introduzca la cadena de operación.");
      printf("\nX + Y =  o  X - Y =... ");
      gets(e);
      while(e[i] != NULL)
      {
            if(e[i] != 32 && !isdigit(e[i]) && e[i] != '+' &&
               e[i] != '-' && e[i] != '=')
            {
            err = 1;
            break;
            }
            else if(e[i] == 32 && (est == 0 || est == 2 ||
                    est == 3 || est == 5)) i++;
            else if(isdigit(e[i]) && (est == 0 || est == 3))
            {
                  est++;
                  n[j] = e[i];
                  i++;
                  j++;
            }
            else if(isdigit(e[i]) && (est == 1 || est == 4))
            {
                  n[j] = e[i];
                  j++;
                  i++;
            }
            else if(e[i] == 32 && (est == 1 || est == 4))
            {
                  n[j] = NULL;
                  if(est == 1) a1 = atol(n);
                  else a2 = atol(n);
                  est++;
```

```
                j = 0;
                i++;
          }
          else if((e[i] == '+' || e[i] == '-') && est == 2)
          {
                op = e[i];
                i++;
                est = 3;
          }
          else if(e[i] == '=' && est == 5)
          {
                printf("%lu %c %lu = ",a1,op,a2);
                printf("%lu", op=='+' ? a1 + a2 : a1 - a2);
                break;
          }
          else
          {
                err = 1;
                break;
          }
    }
    if(err == 1) printf("Error de entrada de datos");
}
```

Ejercicios.

48. ***Escribir un programa que solicite del usuario la entrada de una cadena y muestre por pantalla en número de veces que se ha introducido cada una de las cinco vocales.***

```
#include <stdio.h>
void main(void)
{
    char cadena[100];
    short int a, e, i, o, u, cont;
    printf("Introduzca una cadena de texto ... \n");
    gets(cadena);
    a = e = i = o = u = 0;
    for(cont = 0 ; cadena[cont] != NULL ; cont++)
    {
          if(cadena[cont] == 'a') a++;
          else if(cadena[cont] == 'e') e++;
          else if(cadena[cont] == 'i') i++;
```

```
            else if(cadena[cont] == 'o') o++;
            else if(cadena[cont] == 'u') u++;
      }
      printf("La cadena introducida ha sido ...\n");
      printf("%s\n",cadena);
      printf("Y las vocales introducidas han sido ... \n");
      printf("a ... %hd\n",a);
      printf("e ... %hd\n",e);
      printf("i ... %hd\n",i);
      printf("o ... %hd\n",o);
      printf("u ... %hd\n",u);
}
```

Una sola observación al código: la asignación concatenada que pone a cero las cinco variables de cuenta de vocales. Esta sintaxis es correcta y está permitida en C.

49. ***Escribir un programa que solicite del usuario la entrada de una cadena y muestre por pantalla esa misma cadena en mayúsculas.***

```
#include <stdio.h>
#include <ctype.h>
void main(void)
{
      char cadena[100];
      short int cont;
      printf("Introduzca una cadena de texto ... \n");
      gets(cadena);
      for(cont = 0 ; cadena[cont] != NULL ; cont++)
            cadena[cont] = toupper(cadena[cont]);
      printf("La cadena introducida ha sido ...\n");
      printf("%s\n",cadena);
}
```

50. ***Escribir un programa que solicite del usuario la entrada de una cadena y luego la imprima habiendo eliminado de ella los espacios en blanco.***

```
#include <stdio.h>
#include <string.h>
void main(void)
{
      char cadena[100];
      short int i, j;
      printf("Introduzca una cadena de texto ... \n");
      gets(cadena);
      for(i = 0 ; cadena[i] != NULL ; )
            if(cadena[i] == 32)
                  for(j = i ; cadena[j] != NULL ; j++)
                        cadena[j] = cadena[j + 1];
            else i++;
      printf("La cadena introducida ha sido ...\n");
      printf("%s\n",cadena);
}
```

Si el carácter *i* es el carácter blanco (ASCII 32) entonces no se incrementa el contador sino que se adelantan una posición todos los caracteres hasta el final. Si el carácter no es el blanco, entonces simplemente se incrementa el contador y se sigue rastreando la cadena de texto.

Una observación: la estructura ***for*** situada inmediatamente después de la función *gets*, controla una única sentencia simple, que está controlado por una estructura ***if–else***, que a su vez controla otra sentencia ***for***, también con una sola sentencia simple. No es necesario utilizar ninguna llave en todo ese código porque no hay ni una sola sentencia compuesta.

51. ***Escribir un programa que solicite del usuario la entrada de una cadena y la entrada de un desplazamiento. Luego debe volver a imprimir la cadena con todos los caracteres alfabéticos desplazados tantas letras en el alfabeto como indique el desplazamiento. Si en ese desplazamiento se llega más allá de la letra 'Z', entonces se continúa de nuevo con la 'A'. Se tienen 26 letras en el alfabeto ASCII.***

```
#include <ctype.h>
#include <stdio.h>
void main(void)
{
      char c[100];
      short int i;
      short int d;
      printf("Introduzca una cadena de texto ... \n");
      gets(c);
      for(i = 0 ; c[i] != NULL ; i++)
            c[i] = toupper(c[i]);
      printf("Introduzca el desplazamiento ... \n");
      scanf("%hd",&d);
      d %= 26; /* Si d = 26 * k + d', donde d' < 26
                  entonces desplazar d en el
                  abecedario es lo mismo que desplazar d'. */
      for(i = 0 ; c[i] != NULL ; i++)
      {
            if(isalpha(c[i]))
            {
                  c[i] += d;
                  if(c[i] > 'Z') c[i] = 'A' + c[i] - 'Z' - 1;
            }
      }
      printf("La cadena transformada queda ... \n");
      printf("%s",c);
}
```

Una única observación a este código y, en general, a todos los que se presentan como ejemplo en este manual. Si bien vienen resueltos, cuando de verdad se aprende a programar es en el momento en que uno se pone delante de la pantalla y comienza a echar líneas de código en busca de **SU** solución. Este programa, como la mayoría, tiene infinidad de formas diferentes de solventarlo. La que aquí se muestra no es la mejor, simplemente es la que se ha ocurrido a quien esto escribe en el tiempo que ha tardado en redactar esa solución. Si hubiera resuelto el programa mañana, el libro tendría una solución distinta.

Y es que copiar el código en un editor de C y compilarlo no sirve para aprender. Que este manual es de C, y no de mecanografía.