

En este código, se declara y utiliza el puntero genérico void. Para su utilización efectiva como rvalue en donde el lvalue⁷ no sea otro puntero genérico, los punteros void han de ser convertidos en los punteros del tipo adecuado mediante el operador de moldeado (cast). Esta clase de punteros son muy usados en la gestión dinámica de memoria (ver apartado 3.2).

1.2.4. PUNTEROS A PUNTEROS

Un puntero es también una variable, luego su dirección puede ser almacenada por otra variable puntero; de este modo es posible crear punteros a punteros. Es posible dar una vuelta más de tuerca y generar un puntero a un puntero a puntero, y así sucesivamente. No obstante, se considera que superar el segundo nivel de direccionamiento (punteros a punteros) supone un fuerte decremento de la inteligibilidad del algoritmo asociado, y una posible fuente de errores. De todos modos, veamos un código de ejemplo con punteros de tercer nivel de direccionamiento:

```
#include <stdio.h>
int main()
{ int x, *a, **b, ***c ;      /* 1 */
  a = &x ;                  /* 2 */
  *a = 100 ;                 /* 3 */
  b = &a ;                  /* 4 */
  **b += *a ;               /* 5 */
  c = &b ;                  /* 6 */
  ***c += **b + *a ;       /* 7 */
  /* ... */
}
```

A continuación se analizan las líneas comentadas con un número /* n° */

1. Se declaran: x como una variable de tipo entero, a como un puntero a objetos de tipo entero, b como un puntero a un puntero, el cual a su vez apuntará a objetos de tipo entero. Se dice que b es "puntero de punteros a variables de tipo entero". c como un puntero a un puntero que apunta a otro puntero, el cual a su vez apunta a objetos de tipo entero. Se dice que c es "puntero de punteros de punteros a variables de tipo entero" (obsérvese lo que comentábamos sobre el segundo nivel)
2. Se asigna a a la dirección de x.
3. Se asigna 100 a x; para ello se utiliza indirección a través del puntero a.
4. Se asigna a b la dirección de a.
5. Equivale a $x = x + x$, es decir, $x = 100 + 100 = 200$. Para ello se utiliza un mecanismo de indirección mixto; por un lado doble indirección a través de b, por otro indirección simple a través de a.
6. Se asigna a c la dirección de b.
7. Equivale a $x += x + x = x + x + x$, es decir⁸, $x = 200 + 200 + 200 = 600$. Para ello se utiliza un mecanismo de indirección mixto; por un lado triple indirección a través de c, por otro doble indirección a través de b, y por último indirección simple a través de a.

1.2.5. PUNTEROS Y FUNCIONES

1.2.5.1. Punteros y parámetros funcionales

En C, por defecto, en el paso de parámetros al llamar a funciones, lo que se transfiere es una copia del contenido del parámetro real, es decir un paso por valor. La única salvedad se produce con las matrices, en cuyo caso siempre se pasa la dirección. No obstante, es posible transferir la dirección de una variable, para lo que habrá que emplear el operador dirección (&). Veamos un ejemplo:

```
void f(int v1, int &v2)      /* paso de v1 por valor y v2 por referencia */
{ v1 ++;
  v2 ++;
}
/*...*/
int main ()
{ int x, y;
  /*...*/
}
```

⁷ Los términos lvalue y rvalue provienen de left value y right value (valor izquierdo y valor derecho) en referencia a los dos términos de una sentencia de asignación.

⁸ Recordemos que los operadores de asignación se evalúan de izquierda a derecha.

```
f(x, y); /* aunque aparentemente no hay diferencias, el valor de y cambiará, pero
el de x no*/
/*...*/ }
```

Una manera de no perder de vista que se está usando paso por referencia es utilizar punteros como parámetros formales, que en realidad es el único método permitido por C estándar, ya que el uso del operador & en declaraciones y definiciones de funciones es una innovación de C++. De este modo, el código de la anterior función f quedaría como:

```
void f(int v1, int * v2) /* paso de v1 por valor y un puntero a v2 */
{ v1 ++; /* La sintaxis *v2 ++ evidencia que lo que se está haciendo es variar */
*v2 ++; } /* el contenido del entero cuya dirección es apuntada por v2 */
```

Aunque parezca un contrasentido, en este caso no se está pasando el parámetro por referencia, sino por valor. Lo que ocurre es que se está trabajando con punteros, lo que permite acceder al contenido de la memoria, utilizando indirección, y modificarlo directamente. Sin embargo, y por la razón comentada, el puntero en sí no podrá ser alterado (a menos que se pase un puntero a puntero).

Para la utilización de punteros como parámetros de funciones hay que seguir las siguientes normas:

- Parámetros formales: tanto en la declaración como en la definición de una función capaz de recibir un puntero, éste debe figurar declarado como tal entre los parámetros formales, para ello se acompañará al nombre del puntero de un asterisco *.
- Parámetros actuales: en el momento que se llama a la función, el nombre del puntero debe aparecer solo (sin asterisco). También se puede pasar una dirección para lo que se utilizará el operador dirección & aplicado sobre una variable.

Veamos un ejemplo de funciones que admiten punteros como parámetros (el ejemplo también utiliza conceptos sobre matrices que se estudian en el apartado 2.4.1):

```
#define NELEM 4
float sumaElemMatriz(float *sumandos, int tam); /* parámetros formales (declaración) */
int main()
{ float total, *pfl; /* se declara un puntero a flotante */
static float matrizDatos[NELEM] = {1.34, 2e5, 1200.23, 432.};
pfl = matrizDatos; /* se guarda la dirección de la matriz en pfl */
total = sumaElemMatriz(pfl, NELEM); /* parámetro actual sin asterisco */
printf("\nLa suma de los elementos de matrizDatos es %f", total);
}
float sumaElemMatriz(float *vector, int tamaño) /*parámetros formales (definición)*/
{ int cont;
float acum = 0.;
for (cont = 0; cont < tamaño; cont++) acum += *(vector + cont);
return acum;
}
```

En el ejemplo se observa como se puede pasar a una matriz un puntero. Hay que indicar que la expresión *(vector + cont) es equivalente a utilizar vector[cont] y hace referencia al contenido de matrizDatos[.].

Una función puede retornar un puntero. Para ello tanto en la declaración como en la definición de la función debe aparecer el asterisco.

1.2.5.2. Punteros a funciones

En un programa en ejecución, cada función puede ser localizada por la dirección de memoria a partir de la cual se carga (lo que se suele conocer como *punto de entrada* de la función). De este modo, resulta perfectamente posible utilizar punteros para referenciar funciones, de forma totalmente equiparable a como se hace con las variables, es decir, usar el puntero en lugar del nombre de la función. El uso de punteros permite también el paso de funciones como parámetros de otras funciones, en el que la que es una de las más avanzadas posibilidades de este mecanismo de indirección.

Para crear un puntero a función, se ha de declarar del mismo tipo que el devuelto por la función, seguido de los tipos de los parámetros formales entre paréntesis. En las asignaciones de apuntadores a funciones debe haber una concordancia exacta entre los tipos de los parámetros especificados en la declaración de puntero y los que se indican en la declaración de la función. Además, en la declaración del puntero a una función es preciso utilizar paréntesis alrededor de la variable puntero, ya que el operador de llamada de función, (), tiene un orden de evaluación mayor que el de puntero, *. Veamos un ejemplo:

```
void (*ptr_fun) (int, int);
```

Se está declarando una variable puntero, ptr_fun, que apunta a una función que no devuelve ningún valor (lo que se indica con el tipo devuelto void) y que admite dos parámetros de tipo entero.