Los punteros en C

1 Introducción

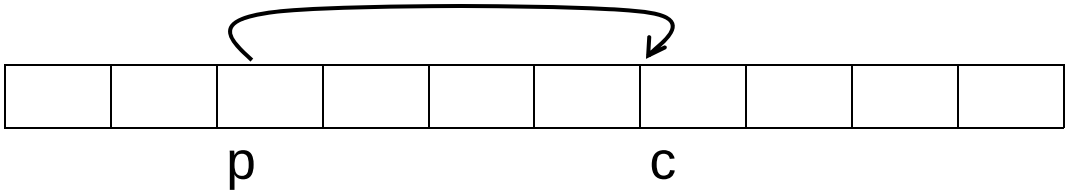
- **¿Cómo se organiza la memoria asociada a un programa?**

Como una colección de posiciones de memoria consecutivas. En ellas se almacenan los distintos tipos de datos, que ocupan, por ejemplo:

1 char = 1 byte
1 int = 2 bytes
1 float = 4 bytes

Un puntero no es más que una variable estática cuyo contenido es una dirección de memoria.

Los punteros, por lo tanto, guardan en dos, cuatro ó n posiciones de memoria, la dirección de un conjunto de celdas.



donde:

p es un puntero a carácter char *p;
c es una variable de tipo carácter char c;

Inicialmente un puntero no apunta a ningún sitio. En C el valor NULL se reserva para indicar que el puntero está vacío (equivalente al nil de la teoría).

- **Operadores asociados a punteros**

&: operador dirección: me da la dirección de un objeto en la memoria.
Sólo sirve para posiciones de memoria (puede apuntar a variables o a vectores, pero no a constantes o expresiones).

Ejemplo:

```
/* Prog1.c */
p = &c;
```

***: operador indirección:** me da el contenido de una posición de memoria (generalmente almacenada en un puntero). Es el equivalente al operador ➔ del código algorítmico:

```
printf ("\nNo tiene lo mismo %c que %ld", c, p);
printf ("\nTiene lo mismo %c que %c", c, *p);
```

Un puntero siempre está asociado a objetos de un tipo: sólo puede apuntar a objetos (variables o vectores) de ese tipo.

```
int *ip;                /* Sólo puede apuntar a variables enteras */
char *c;                /* Sólo puede apuntar a variables carácter */
double *dp,             /* dp sólo puede apuntar a variables reales */
float atof (char *);    /* atof() es una función que devuelve un real dada una
                           cadena que se le pasa como puntero a carácter*/
```

Los operadores * y & son unarios y tienen más prioridad a la hora de evaluarlos que los operadores binarios.

/* Mirar **prog2.c***/

```
*ip = *ip + 10;     printf ("\n y=%d, *ip=%d, ip=%ld", y, *ip, ip);
y = *ip + 10;       printf ("\n y=%d, *ip=%d", y, *ip);
*ip += 1;           printf ("\n y=%d, *ip=%d", y, *ip);
```

Es necesario utilizar paréntesis cuando aparezcan en la misma expresión que otros operadores unarios como ++ o --, ya que en ese caso se evaluarían de izquierda a derecha

```
(*ip)++;            printf ("\n y=%d, *ip=%d", y, *ip);
*ip++;              printf ("\n y=%d, *ip=%d", y, *ip);
```

Dado que los punteros son variables, también pueden usarse como asignaciones entre direcciones. Así:

```
int *ip, *iq;
iq = ip;
```

Indica que iq apunta a donde apunte el puntero ip.

2. Los punteros y los argumentos a funciones

Código Algorítmico	Equivalente C	Ejemplo
parámetro dato	paso por valor	int scanf (**const char ***, <arg>)
parámetro resultado	paso por referencia / valor devuelto por una función	**int** scanf (const char *, <arg>)
parámetro dato-resultado	paso por referencia	scanf (“%d”, **&entero**);

En C, por defecto, todos los parámetros a las funciones se pasan por valor (la función recibe una copia del parámetro, que no se ve modificado por los cambios que la copia sufra dentro de la función).

Ejemplo: Intercambio de dos valores {véase **uso_punteros+intercambia.c**}

{VERSION ERRONEA}
intercambia_mal (int a, int b) {
int tmp;
tmp = a;
a = b;
b = tmp;}

{VERSION CORRECTA}
intercambia_bien (int *a, int *b) {
int tmp;
tmp = *a;
*a = *b;
*b = tmp; }

Para que un parámetro de una función pueda ser modificado, ha de pasarse por referencia, y en C eso sólo es posible pasando la dirección de la variable en lugar de la propia variable.

Si se pasa la dirección de una variable, la función puede modificar el contenido de esa posición (no así la propia dirección, que es una copia).

3 Punteros y vectores

En C los punteros y los vectores están fuertemente relacionados, hasta el punto de que **el nombre de un vector es en sí mismo un puntero a la primera (0-ésima) posición del vector**. Todas las operaciones que utilizan vectores e índices pueden realizarse mediante punteros.

int v[10];

1	3	5	7	9	11	13	15	17	19
v[0]	V[1]	v[2]	v[3]	v[4]	v[5]	v[6]	v[7]	v[8]	v[9]

int v[10]; /* designa 10 posiciones consecutivas de memoria donde se pueden almacenar enteros */
int *ip; **/*** Designa un puntero a entero */

ip, es un puntero y, por lo tanto, permite acceder a una posición de memoria. Sin embargo, como se ha dicho antes **v** también puede considerarse como un puntero a entero, de tal forma que las siguientes expresiones son equivalentes:

ip = &v[0]	**ip = v**
x = *ip;	**x = v[0];**
***(v + 1)**	**v[1]**
v + i	**&v[i]**

4 Aritmética de punteros

El compilador C es capaz de “adivinar” cuál es el tamaño de una variable de tipo puntero y realiza los incrementos/decrementos de los punteros de la forma adecuada. Así:

```
int v[10], *p;
p = v;

/* p          Apunta a la posición inicial del vector*/
/* p + 0      Apunta a la posición inicial del vector*/
/* p + 1      Apunta a la segunda posición del vector*/
/* p + i      Apunta a la posición i+1 del vector */

p = &v[9];  /* p apunta ahora a la última posición (décima) del vector */

/* p - 1      Apunta a la novena posición del vector*/
/* p - i      Se refiere a la posición 9 - i en v*/
```

Ejemplo de recorrido de un vector utilizando índices y punteros.

```
/* prog3.c (revisar la versión extendida de este programa)*/

main(){
int v[10];
int i, *p;

for (i=0; i < 10; i++) v[i] = i; /* Asignamos valores iniciales*/
for (i=0; i < 10; i++) printf ("\n%d", v[i]);

p = v;
for (i=0; i < 10; i++) printf ("\n%d", *p++);
/* Tras cada p++ el puntero señala a la siguiente posición en v */
}
```

C realiza las mismas operaciones con independencia del tipo de la variable.

De esta forma, si tenemos dos variables que son punteros:

```
int *ip, vi[10];        /* Necesita dos posiciones para representar un entero */
char *cp, vc[10];       /* Necesita una posición para representar un carácter */

ip = &vi[0];
cp = &vc[0];
```

Las expresiones:

```
ip + 1, ó ip++      Apunta al siguiente elemento (dos posiciones): vi[1]
cp + 1, ó cp++      Apunta al siguiente elemento (una posición): vc[1]
```

Diferencia entre puntero y nombre de vector:

Mientras que un puntero es una variable y puede intervenir en asignaciones o incrementos, el nombre del vector no es una variable, por lo tanto, no pueden realizarse operaciones del tipo:

```
int v[10], *a;
v = a; /* no permitido */
v++; /* no permitido */
```

Dado que el nombre del vector es un puntero, puede utilizarse para pasar un vector como parámetro de una función (lo que permite modificar el contenido de esas posiciones de memoria adyacentes, o lo que es lo mismo, los elementos del vector).

```
char palabra[10];

scanf ("%s", palabra);   /* Modifica las posiciones de memoria */
sort (palabra, 0, strlen(palabra));
```

/* **sort** va a modificar el contenido del vector, ordenándolo por cualesquiera de los métodos que se presenten en clase; por lo tanto se pasará la dirección del primer elemento = palabra = &palabra[0].
strlen es una función de la biblioteca estándar, cuyo macro es: int strlen (char *)
La función nos devuelve la longitud de un vector de caracteres */

Recordatorio: Los vectores de caracteres terminan en el carácter '\0'.

Ejemplo 1: Realícese una función que sea equivalente a **strlen**.

int longitud (char * s)

Ejemplo 2: Realizar una función equivalente que permita realizar a := b, cuando ambas son cadenas.

copia (char * a, char * b)