Ciencias computacionales

Propedéutico: Programación

Contents

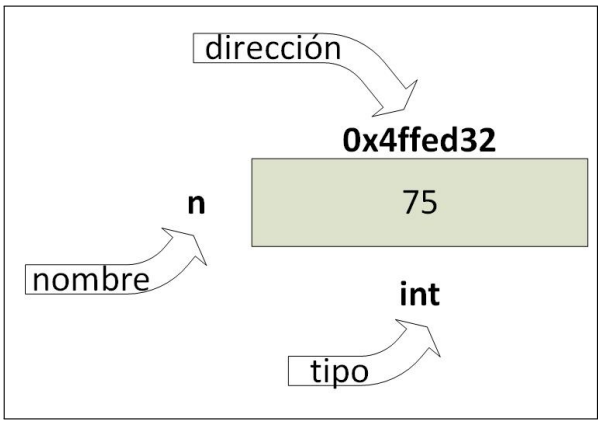

Figure 1: Dirección de memoria de una variable

1 Apuntadores

1.1 Vídeos sobre apuntadores

Video sobre pilas.
Video sobre colas.

1.2 Definición

Un puntero es una herramienta que se utiliza para hacer programas flexibles y eficientes. Una *variable puntero* contiene direcciones de otras variables, es decir, contiene valores que son direcciones de memoria donde se almacenan datos [1].

En resumen un apuntador es un tipo de dato que "apunta" a otro valor almacenado en memoria.

1.2.1 Direcciones de memoria

De forma gráfica cuando una variable se declara en una dirección de memoria su nombre, su tipo. Como se muestra en la Fig. 1.

A la dirección de la variable se puede acceder por medio del operador de dirección &. Como ejemplo se puede imprimir la dirección de n con la siguiente sentencia:

$printf(\text{“}\%p\text{”}, \&n)$

Se puede ganar en eficacia en el acceso a esta dirección de memoria utilizando un apuntador.

El concepto de apuntadores tiene correspondencia con la vida cotidiana. Por ejemplo, cuando se envía una carta por correo, su información se entrega gracias a su apuntador que es la dirección de esa carta.

Los apuntadores se rigen por las siguientes reglas:

- Un *apuntador* es una variable como cualquier otra.
- Una *variable* apuntador contiene una *dirección* que apunta a otra posición en memoria.
- En esa *posición* se almacenan los datos a los que apunta el apuntador.
- Un apuntador apunta a una variable de memoria.

El ejemplo que se muestra en la Fig. 2 presenta tanto el código en C, como de manera gráfica lo que sucede cuando se asigna un apuntador.

1.2.2 Declaración de apuntadores

Para declarar un apuntador se debe seguir el formato:

$< tipodedatoapuntado > * < identificadordelpuntero >$

Algunos ejemplos:

Ejemplo

```
#include<stdio.h>
Void main (){
    int n = 75 ;
    int *p = &n  /* p variable puntero, tiene la dirección de n*/
    printf("n= %d,  &n = %p, p = %p", n, &n, p) ;
    printf("&p = %p\n", &p) ;
}
```

Ejecución

```
n = 75, &n = 0x4fffd32, p= 0x4fffd32
&p = 0x4fffd30
```

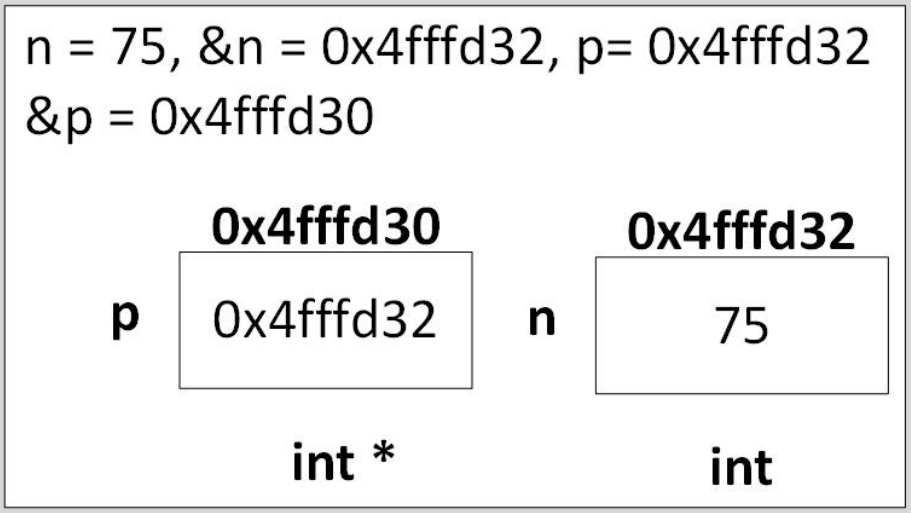

Figure 2: Apuntador a una variable

$int * ptr1$ /*Puntero a un tipo de dato entero (int)*/
$long * ptr2$ /*Puntero a un tipo de dato entero largo (long int)*/
$char * ptr3$ /*Puntero a un tipo de dato char */

1.2.3 Indirección de apuntadores

El *indireccionar un apuntador* conlleva obtener el valor al que esta apuntando, esto se logra utilizando el operador de indirección * [1].
$int \quad edad$;
$int * p\_edad$;
$p\_edad = \&edad$
$p\_edad = 50$;

El código anterior se muestra de manera gráfica en la Fig. 3. Si se desea imprimir el valor de edad, se puede hacer de dos formas distintas, una imprimiendo directamente el valor de edad y la otra desreferenciando al apuntador edad como se muestra a continuación, respectivamente según el caso.
$printf(\text{“}\%d\text{”}, edad)$;
$printf(\text{“}\%d\text{”}, *p\_edad)$;

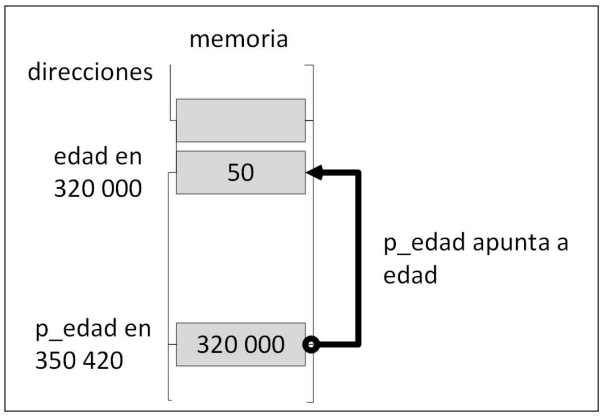

Figure 3: Indirección

1.2.4 Apuntadores NULL y void

Un *apuntador nulo (NULL)* no apunta a ningún dato valido. Sirve a los programadores para saber cuando un apuntador no direcciona a un dato válido. Se pueden utilizar librerías en las cuales este apuntador esta declarado (stdio.h, stdlib.h y string.h) o se puede definir NULL en la parte superior del programa.

$\#define \quad NULL \quad 0$

Por otro lado, existe la noción de apuntador que apunta a cualquier tipo de dato, no se asigna algún tipo de dato específico. Esto se logra mediante un apuntador *void**, denominado apuntador genérico.

$void \quad *ptr$; /*declara un apuntador genérico*/

Los apuntadores void pueden direccionar a una variable char, float o una cadena, por nombrar algunos casos.

1.2.5 Apuntadores a apuntadores

Un apuntador puede apuntar a otro apuntador, esto se hace precediendo a la variable con dos asteriscos. La Fig. 4 muestra un ejemplo de apuntadores a apuntadores.

1.3 Arreglos y Apuntadores

Se pueden direccionar arreglos como si fueran apuntadores y apuntadores como si fueran arreglos. Esto implica que se pueden almacenar cadenas de datos en elementos de arreglos. Sin apuntadores esto no es posible, ya que no existe el tipo de dato cadena (string) en C.

1.3.1 Nombres de arreglos como apuntadores

Un nombre de un arreglo es simplemente un apuntador. Supóngase la siguiente declaración de un arreglo:

int lista[5] = {10, 20, 30} ;

Si se manda visualizar *lista[0]* se verá *10*. De igual forma si se manda a visualizar **lista*, también se verá 10. Esto significa que:

lista + 0 *apunta a* lista[0]
lista + 1 *apunta a* lista[1]
lista + 2 *apunta a* lista[2]

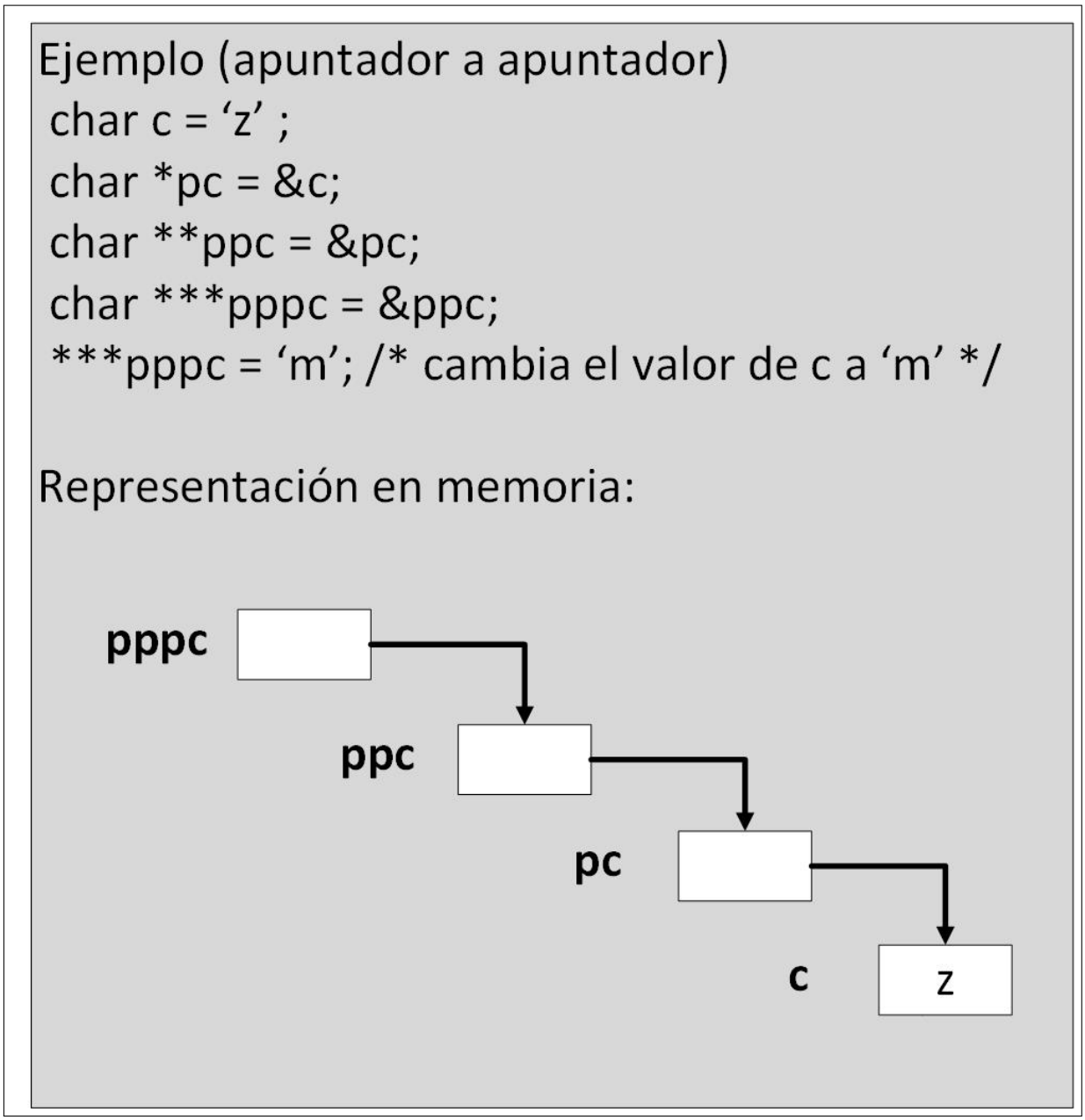

Figure 4: Apuntador a apuntador

El nombre de un arreglo es un apuntador, contiene la dirección en memoria de comienzo de la secuencia de elementos que forman el arreglo. Un arreglo es un apuntador constante ya que no se puede modificar, sólo se puede acceder par indexar a los elementos del arreglo.

1.3.2 Arreglos de apuntadores

Un arreglo de apuntadores es un arreglo que contiene como elementos apuntadores, cada uno de ellos apunta aun tipo de dato específico. Cada elemento contiene una dirección que *apunta* a otros valores de la memoria. Por ejemplo:
int *ptr[10] /* reserva un array de 10 apuntadores enteros */

Se puede asignar un elemento de *ptr* una dirección, por ejemplo:
ptr[5] = &edad /* ptr[5] apunta a la dirección edad */

ptr [4] = NULL; /* ptr[4] no contiene dirección alguna*/

1.3.3 Aritmética de apuntadores

Una variable apuntador es una variable que se puede modificar. A un apuntador se l e puede sumar o restar un entero n; esto hace que apunte n posiciones adelante o atrás del valor actual. También se les puede aplicar el operador $++$ o el operador $--$. Haciendo que contenga la dirección siguiente o anterior del elemento. Por el contrario no tiene sentido, sumar o restar una constante de coma flotante.

1.3.4 Apuntadores como argumentos de funciones

Cuando se pasa un argumento por *valor* no se puede cambiar el valor de esa variable. Pero si se pasa un apuntador de una variable a una función (por *refrencia*), se puede modificar el valor de dicha variable. Una variable local a una función se puede hacer visible a otra función pasándola como argumento, si se desea cambiar el valor de dicha variable se utilizan los apuntadores.

Ejemplo:

```
void Incrementar(int *i){
    *i +=5 ;
}
```

Para llamar a la función es necesario el paso de una dirección de memoria:

```
int i = 10 ;
Incrementar(&i) ;
```

También es posible que los apuntadores se utilicen para direccionar funciones, estos apuntan al código ejecutable, debido que las funciones también radican en algún espacio en memoria. La sintaxis general es la siguiente:

```
Tipo_de_retorno (*PunteroFuncion) (lista de parámetros) ;
```

Por ejemplo:

```
int f(int) ; /* Declara la función f */
int (*pf) (int) ; /* Define puntero pf a función int con argumento int */
pf = f ; /* asigna la dirección de f a pf*/
```

1.4 Asignación dinámica

La asignación dinámica de memoria es necesaria cuando no es posible conocer cuanta memoria se debe asignar a una variable, ya que aveces se reserva más espacio del necesario, lo cual conlleva a problemas de eficiencia. Para asignar memoria en tiempo de ejecución se hace mediante las funciones *malloc(), calloc() y free ()*.

1.4.1 Malloc ()

Malloc () asigna un bloque de memoria, es decir, el número de bytes pasados como argumento a la función *malloc()*. Esta función a su vez devuelve un apuntador (*void**) a dicho bloque de memoria. La sintaxis para el uso de esta función es la siguiente:

```
apuntador = malloc (tamño en bytes) ;
```

Ejmeplo:

```
long *p;
p = (long *) malloc (32*sizeof(long) ;
```

Si no existe un espacio de almacenamiento suficiente, la función malloc() devuelve NULL, la escritura de un programa totalmente seguro exige comprobar el valor devuelto por malloc() para asegurar que no sea NULL.

1.4.2 Free ()

Para liberar el espacio de memoria previamente creado por la función *malloc ()* se utiliza la función *free ()*. Esto con el fin de garantizar que siempre existirá memoria disponible para asignar otros bloques de memoria. El formato es el siguiente:

free (apuntador)
    Ejemplo:
int *ad;
ad = (int*) malloc(sizeof(int)) ;
    Para liberar el espacio de memoria se utilizaría
free (ad) ;

References

[1] Noel Kalicharan. *Advanced Topics in C: Core Concepts in Data Structures.* Apress, Berkely, CA, USA, 1st edition, 2013.