Introducción a Punteros

Pablo R. Ramis

Universidad Nacional de Rosario, Instituto Politécnico, Dto. de Informática,
prramis@ips.edu.ar,
WWW home page: http://informatica.ips.edu.ar

Resumen un puntero es un objeto del lenguaje de programación, cuyo valor se refiere a (o apunta a) otro valor almacenado en otra parte de la memoria del ordenador utilizando su dirección. Un puntero referencia a una ubicación en memoria, y a la obtención del valor almacenado en esa ubicación se la conoce como desreferenciación del puntero.

1. Memoria y puntero

La memoria de la computadora no deja de ser una colección de espacios donde se almacenan datos. Estos datos varian su tamaño y por lo tanto la porción de memoria que usan dependiendo el tipo que sean: 1 char es 1 byte, un int son 2 bytes, etc.

Un puntero no deja de ser una variable más la cual guarda una dirección de memoria, en ella se encuentra otra variable (y su contenido) que deberá tener que ser del tipo que se declaró el puntero. Ej:

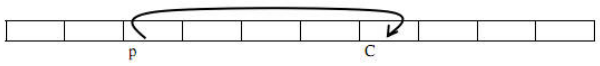

Figura 1. variables en memoria

donde: pc es un puntero a caracter y c es una variable de tipo char
Inicialmente un puntero no apunta a ningún sitio, o sea su valor es NULL.

```
1  /*programa1.c*/
2  #include <stdio.h>
3
4  int main(){
5      char c, *pc;
6
7      c = 'a';
8      pc = &c; /* asigno al puntero la direccion de memoria de
           c */
9
```

```
10      printf("\nEl contenido de cada uno es: c #%c# y en pc #%d
            #\n", c, pc);
11      printf("\nEl contenido es igual c #%c# y *pc #%c#\n", c,
            *pc);
12      printf("\nApuntan al mismo sitio &c #%d# y pc #%d#\n", &c
            , pc);
13
14      return 0;
15  }
```

Al compilar, veremos una advertencia en la cual se nos dice que estamos mostrando en forma equivocada a pc. Esto es porque buscamos mostrar la dirección de la memoria como entero. Ignoraremos esas advertencias.

```
$ gcc -o programa1 programa1.c
programa1.c: In function ‘’main:
programa1.c:10:5: warning: format ‘%’d expects argument of type ‘’int, but
     argument 3 has type ‘char ’* [-Wformat=]
      printf("\nEl contenido de cada uno es: c #%c# y en pc #%d#\n", c, pc);
      ^
programa1.c:12:5: warning: format ‘%’d expects argument of type ‘’int, but
     argument 2 has type ‘char ’* [-Wformat=]
      printf("\nApuntan al mismo sitio &c #%d# y pc #%d#\n", &c, pc);
      ^
programa1.c:12:5: warning: format ‘%’d expects argument of type ‘’int, but
     argument 3 has type ‘char ’* [-Wformat=]
$
$./programa1

El contenido de cada uno es: c #a# y en pc #1822314359#

El contenido es igual c #a# y *pc #a#

Apuntan al mismo sitio &c #1822314359# y pc #1822314359#
$
```

1.1. Operadores

Cuando necesitamos conocer la dirección de una variable o arreglo utilizaremos el operador *&*. Si nos fijamos en el ejemplo del programa1 vemos en la línea 8 que se está asignando a pc la dirección de c.

Y si tenemos un puntero y quiero ver el contenido de la dirección de la memoria a la que apunta utilizaremos ***** como se ve en la línea 11 del código.

Estos dos operadores son unarios y tienen prioridad a la hora de evaluarlos con operadores binarios. Ejemplo:

```
1   /*programa2.c*/
2
3   #include <stdio.h>
4
5   int main(){
6       int y, *ip;
```

```
7
8       y = 12;
9       ip = NULL;
10
11      printf ("\nValor de y #%d#, de ip #%d# sin asignar\n", y,
            ip);
12
13      ip = &y;
14
15      printf ("\nip = &y: Valor de y #%d#, su direccion &y #%d#
            y de ip #%d#\n",
16              y, &y, ip);
17
18      printf ("\nValor de y #%d#, y de *ip #%d#\n", y, *ip);
19
20      *ip = *ip + 10;
21      printf ("\n*ip = *ip + 10: : Valor de y #%d#, de *ip #%d
            #\n", y, *ip);
22
23      y = *ip + 10;
24      printf ("\ny=*ip + 10: Valor de y #%d#, de *ip #%d#\n", y
            , *ip);
25
26      *ip += 1;
27      printf ("\n*ip += 1: Valor de y #%d#, de *ip #%d#\n", y,
            *ip);
28
29      printf ("\n(*ip)++: Valor de y #%d#, de *ip #%d# en ese
            momento\n", y, (*ip)++);
30      printf ("\nValor de y #%d#, de *ip #%d# despues\n", y, *
            ip);
31
32      printf ("\n+++ip: Valor de y #%d#, de *ip #%d# en este
            momento\n", y, +++ip);
33      printf ("\nValor de y #%d#, de *ip #%d# despues\n", y, *
            ip);
34
35      return 0;
36  }
```

```
$ gcc -o programa2 programa2.c
```

Al igual que en el caso anterior, ignoraremos los warnings de la compilación.

```
$ ./programa2

Valor de y #12#, de ip #0# sin asignar

ip = &y: Valor de y #12#, su direccion &y #954195124# y de ip #954195124#
```

```
Valor de y #12#, y de *ip #12#

*ip = *ip + 10: : Valor de y #22#, de *ip #22#

y=*ip + 10: Valor de y #32#, de *ip #32#

*ip += 1: Valor de y #33#, de *ip #33#

(*ip)++: Valor de y #34#, de *ip #33# en ese momento

Valor de y #34#, de *ip #34# despues

++*ip: Valor de y #35#, de *ip #35# en este momento

Valor de y #35#, de *ip #35# despues
```

Como comentario final podemos, considerando que los punteros son variables, es posible lo siguiente

```
1
2       int *pc1, *pc2;
3
4       pc1 = pc2;
```

2. Punteros y arrays

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]

Figura 2. arreglo de 10 posiciones

v: designa 10 posiciones consecutivas de memoria donde se pueden almacenar enteros.

int *ip;

Designa un puntero a entero, por lo tanto puede 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:

```
1
2   ip = &v[0];
```

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

3. Aritmética de punteros

Con la variable de tipo puntero al incrementar o decremntar tendríamos un desplazaminto por la memoria que nos permitiría acceder a sus contenidos. Entonces:

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*/
Veamos un ejemplo completo:

```
 /*programa3.c*/

#include <stdio.h>
main(){
    int v[100];
    int i, *p;

    for (i=0; i < 8; i++)
        v[i] = i;

    printf("\nRecorrido usando v[i]:\n");

    for (i=0; i < 8; i++)
        printf ("%d\t", v[i]);

    printf ("\n");
    printf("\nRecorrido usando punteros:\n");

    p = v;

    for (i=0; i < 8; i++)
```

```
22          printf ("%d\t", *p++); /* equivale a mostrar *p y
                hacer p++*/
23
24      printf("\n");
25      printf ("\nAhora calculo las direcciones y muestro el
            contenido\n");
26
27      p = v;
28
29      for (i=0; i < 8; i++)
30          printf ("%d\t", *(p+i));
31
32      printf("\n");
33
34  /* Calcula la ódireccin, por los éparntesis, y édespus
        muestra el contenido */
35  /* Tras cada p++ el puntero ñseala a la siguiente óposicin en
         v */
36
37  /* DIFERENCIA ENTRE *p++ y (*p)++*/
38
39      printf ("\n DIFERENCIA ENTRE *p++ y (*p)++ \n");
40      printf ("\nAhora voy a recorrer el vector utilizando (*p)
            ++\n");
41
42      p = v;
43
44      for (i=0; i < 8; i++)
45          printf ("%d\t", (*p)++); printf("\n");
46
47      printf ("\nAhora voy a recorrer el vector utilizando *(p+
            i)\n");
48
49      p = v;
50
51      for (i=0; i < 8; i++)
52          printf ("%d\t", *(p+i));
53
54      printf("\n");
55
56      printf ("\nAhora voy a recorrer el vector utilizando p[i
            ])\n");
57
58      p = v;
59
60      for (i=0; i < 8; i++)
61          printf ("%d\t", p[i]);
62
63      printf ("\nComo se ve, los ultimos dos son lo mismo.\n");
64
```

```
65
66  }
```

Compilamos y probamos.

```
$ gcc -o programa3 programa3.c
$ ./programa3

Recorrido usando v[i]:
0       1       2       3       4       5       6       7

Recorrido usando punteros:
0       1       2       3       4       5       6       7

Ahora calculo las direcciones y muestro el contenido
0       1       2       3       4       5       6       7

 DIFERENCIA ENTRE *p++ y (*p)++

Ahora voy a recorrer el vector utilizando (*p)++
0       1       2       3       4       5       6       7

Ahora voy a recorrer el vector utilizando *(p+i)
8       1       2       3       4       5       6       7

Ahora voy a recorrer el vector utilizando p[i])
8       1       2       3       4       5       6       7
Como se ve, los ultimos dos son lo mismo.
```

3.1. Diferencia entre un puntero y el nombre de un array

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; /*Está mal!!!*/
v++; /*Está mal!!!*/