```
/*********************************************************************
 * pruebasConPunteros.c
*********************************************************************/
//
//  tipo nombre     Define una variable (una posición de memoria)
//                  para albergar un dato del tipo indicado.

//  tipo* pNombre   Define un puntero hacia una posición de memoria
//                  donde hay un dato del tipo indicado
//

//  *pNombre        Devuelve el dato hacia el que apunta el puntero.
//     (leer * como "el valor en…")           ("convierte" un puntero en variable)

//  &nombre         Devuelve la posición de memoria donde esta albergado el dato.
//     (leer & como "la dirección de…")      ("convierte" una variable en puntero)
//
//
//
//  Si el tipo es un dato compuesto, un 'struct',
//  para acceder a cada uno de sus datos componentes internos:
//
//   nombre.componente
//   pNombre->componente
//
//

#include <stdio.h>

struct Posicion
{
    int x;
    int y;
    int z;
};


int main(int argc, char **argv)
{
  // con variables simples
    int cantidad;
    cantidad = 5;
    int* pCantidad;
    pCantidad = &cantidad;
    printf("%d , %d", cantidad, *pCantidad);
    printf("\n");
    printf("%p , %p", &cantidad, pCantidad);
    printf("\n\n");

  // con un array de elementos simples
    int cantidades[4];
    cantidades[0] = 10;
    cantidades[1] = 20;
    cantidades[2] = 30;
    cantidades[3] = 40;
    printf(" : ");
    for(int i = 0; i < 4; i++)
    {
        printf("%d : ", cantidades[i]);
    }
    printf("\n");

    int* pCantidades;
    pCantidades = cantidades;  // el propio nombre del array es un puntero a la primera
                               posición

    // estas tres sentencias son equivalentes
    printf("%d", cantidades[0]);
    printf("\n");
    printf("%d", *cantidades);
    printf("\n");
    printf("%d", *pCantidades);
```

```
    printf("\n");

    // esta manera de recorrer el array usando aritmetica de punteros,
    // es equivalente a la de usando indices de array
    printf(" : ");
    for(int i = 0; i < 4; i++)
    {
        printf("%d : ", *(pCantidades + i));
    }
    printf("\n\n");


  // con un array de elementos struct
    struct Posicion puntos[5];
    puntos[0].x = 1;
    puntos[0].y = 2;
    puntos[0].z = 3;
    puntos[1].x = 11;
    puntos[1].y = 12;
    puntos[1].z = 13;
    puntos[2].x = 21;
    puntos[2].y = 22;
    puntos[2].z = 23;
    puntos[3].x = 31;
    puntos[3].y = 32;
    puntos[3].z = 33;
    puntos[4].x = 41;
    puntos[4].y = 42;
    puntos[4].z = 43;

    printf(" : ");
    for(int i = 0; i < 5; i++)
    {
        printf("[%d, %d, %d] : ", puntos[i].x, puntos[i].y, puntos[i].z);
    }
    printf("\n");


    struct Posicion* pPuntos;
    pPuntos = puntos;

    // estas dos sentencias son equivalentes
    printf("[%d, %d, %d]\n", pPuntos[0].x, pPuntos[0].y, pPuntos[0].z);
    printf("[%d, %d, %d]\n", pPuntos->x, pPuntos->y, pPuntos->z);
    // estas dos sentencias son equivalentes
    printf("[%d, %d, %d]\n", pPuntos[2].x, pPuntos[2].y, pPuntos[2].z);
    printf("[%d, %d, %d]\n", (pPuntos + 2)->x, (pPuntos + 2)->y, (pPuntos + 2)->z);
    printf("\n\n");
    // de ambas, la primera usa indexamiento y la segunda usa aritmética de punteros.

    return 0;
}



/*********************************************************
* Este programa muestra este resultado en pantalla:
*
5 , 5
000000000061FDF4 , 000000000061FDF4

 : 10 : 20 : 30 : 40 :
10
10
10
 : 10 : 20 : 30 : 40 :

 : [1, 2, 3] : [11, 12, 13] : [21, 22, 23] : [31, 32, 33] : [41, 42, 43] :
[1, 2, 3]
[1, 2, 3]
[21, 22, 23]
[21, 22, 23]
*
*********************************************************/
```