Punteros

Introducción

La memoria asociada a un programa se organiza como una colección de memorias consecutivas formadas por un conjunto de bytes. En ella se almacenan los distintos tipos de datos, que ocupan:

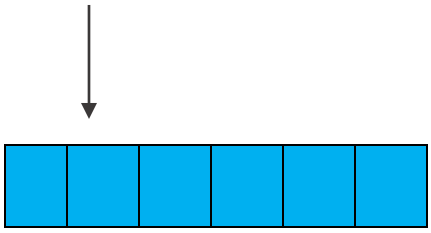
char -> 1 byte

Int - > 2 bytes

Float -> 4 bytes

Cada byte tiene un número asociado, **una dirección en esa memoria.**

Contenido de cada posición de memoria



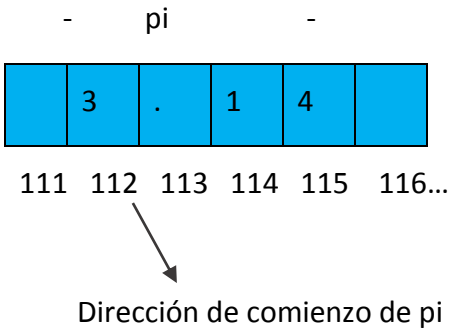
111 112 113 114 115 116… Direcciones de memoria

Veamos un ejemplo:

```
int  main()
{
        float pi; // reserva memoria para la variable pi
        pi=3.14;  // almacena  o guarda el valor 3.14
        printf ("%.2f\n", pi);  // muestra el contenido de pi
        return 0;
}
```

Con la declaración de variable, la instrucción reserva memoria para la variable pi. Un decimal de tipo float ocupa 4 bytes. Dependiendo de la arquitectura del computador, la cantidad de bytes requerida para cada tipo de datos puede variar.

Estado de la memoria después de la declaración de la variable pi:



Cuando se muestra pi, ocurren dos pasos diferenciables. En primer lugar, el programa busca la dirección de memoria reservada para pi (en nuestro ejemplo sería la dirección 112). Hecho esto, en segundo lugar, el programa recupera el contenido de esa dirección de memoria. Así, por un lado distinguimos la dirección de memoria asignada a una variable y, por el otro, el contenido de la posición de memoria reservada.

Podemos acceder a la dirección de una variable utilizando el operador &, y así accedemos a la dirección de memoria de una variable. Esta dirección es un número en sistema hexadecimal con el que podemos trabajar directamente.

Concepto

Como su nombre lo indica, un puntero es algo que apunta, es decir, nos indica la ubicación de algo. Imaginemos que tenemos un gran organizador en el que guardamos informes. Este organizador está dividido en compartimentos, cada uno de los cuales contiene uno de nuestros informes (esto sería equivalente a las variables con las que hemos trabajado hasta ahora -informes- que contienen información, y el organizador representa la memoria; obviamente las variables se almacenan en la memoria). Sin embargo, otros compartimentos no contienen informes sino que una nota que nos dice dónde está ese informe.

Supongamos que, como máximo, trabajamos con cinco informes a la vez. Por lo tanto, reservamos cinco compartimentos en los que indicamos en qué compartimentos se encuentran esos cinco informes. Estos cinco compartimentos serían nuestros punteros y como ocupan un compartimento en el organizador (nuestra memoria) son realmente variables, pero muy especiales. Estas variables punteros ocupan siempre un tamaño fijo y contienen el número de compartimento en el que se encuentra el inicio de la información; no contienen la información en sí.

Así, en nuestro supuesto de que solo trabajemos con cinco informes a la vez, dispondríamos de cinco compartimentos en los que indicaríamos dónde se encuentran esos informes que buscamos y, de esta forma, cuando terminemos con ellos y deseemos trabajar con otros, solo tendremos que cambiar el contenido de esos cinco compartimentos indicando dónde se encuentran los nuevos informes.

Esto es lo que en programación se conoce como **referencia indirecta o indirección**: accedemos a la información a través de un puntero que nos dice dónde se encuentra. A grandes rasgos, los punteros son referencias indirectas a datos en la memoria del ordenador. En C, los punteros son muy importantes puesto que su utilización es básica para la realización de numerosas operaciones, entre ellas: paso de parámetros que deseamos modificar, tratamiento de estructuras dinámicas de datos (variables que no se declaran en el programa y se crean durante la ejecución del programa), cadenas de caracteres…

Definición de puntero

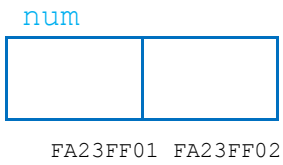
En el famoso libro de Kernighan y Ritchie *El lenguaje de programación C* se define un puntero como “una variable que contiene la dirección de una variable”.

Declaración de un puntero

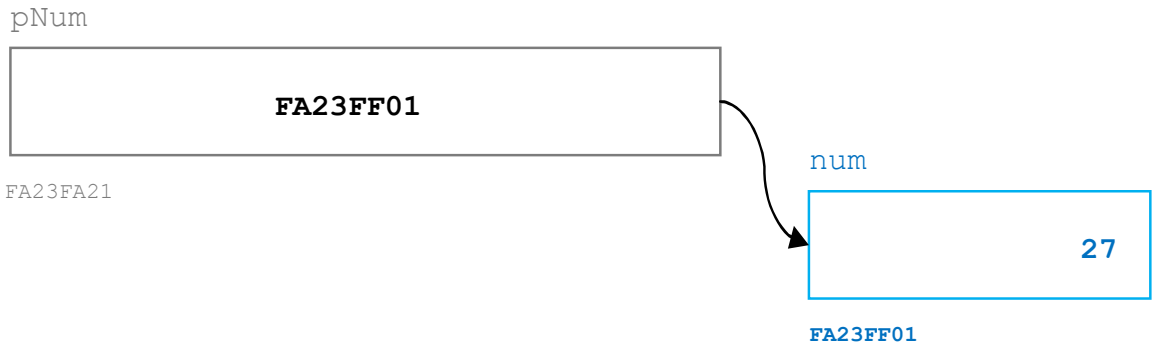
En su declaración, delante de su nombre debe añadirse un asterisco y especificarse el tipo de dato al que apunta.

```
Tipo de dato al que apunta  *nombre_de_variable;
```

Por ejemplo:



Un puntero podrá guardar la dirección donde comienza la variable num (FA23FF01):



int *pnum;

El asterisco delante del nombre de la variable indica que no es una variable estándar sino un puntero. Además, indica que apuntará a un valor de tipo int o a una variable de tipo int. Todos los punteros ocupan 4 bytes pues todos guardan una dirección de memoria.

Ejemplo:

```
float precio, *punteroreal;
```

En este ejemplo, se declara una variable estándar precio que contendrá valores reales (tipo float) y, después, un puntero llamado puntero real que apuntará a valores reales o a una variable de tipo real.

El operador & permite obtener la dirección de memoria de una variable. Así, si queremos que el puntero puntero real contenga la dirección de memoria de la variable real precio, debemos escribir la línea:

```
punteroreal = &precio;
```

Se dice que “el **puntero real** apunta a la variable real precio” cuando contiene la dirección de memoria de la variable. Si un puntero contiene una dirección de memoria está apuntando al dato guardado en dicha dirección. Para poder acceder a él, debe usarse el operador asterisco delante del puntero.

Ejemplo:

```
int main()
{
    int edad; // Declaramos una variable entera.
    int *puntero; // Declaramos un puntero a un valor entero.
    edad=45; // Almacenamos en la variable el valor entero 45.
   puntero=&edad;  // Almacenamos en el puntero la dirección de memoria de la variable edad (usando &).
/* Se dice que “puntero apunta a edad”. Ahora desde el puntero se puede obtener el valor almacenado en edad usando el asterisco. */
// Ahora mostramos el valor almacenado en la dir. de memoria a la que apunta el puntero (usando *).
   printf(“El valor almacenado en la variable es: %d”, *puntero);
// Aparecerá en pantalla: El valor almacenado en la variable es: 45.
   return;
}
```

Es importante no confundir el asterisco que se antepone al nombre del puntero al declararlo para indicar que es un puntero (en la zona de declaraciones de variables), y el asterisco antepuesto al nombre de un puntero dentro del código del programa para indicar que se quiere obtener el dato almacenado en la dirección de memoria a la que apunta el puntero.

Cuando un puntero apunta a una variable, puede obtenerse el valor almacenado en dicha variable usando el operador asterisco, pero también se puede modificar dicho valor.

Ejemplo:

```
int main()
{
   int edad; // Declaramos una variable entera.
   int *puntero; // Declaramos un puntero a un valor entero.
   edad=45; // Almacenamos en la variable el valor entero 45.
   puntero=&edad; // Almacenamos en el puntero la dirección de memoria de la variable edad (usando &).
/* Se dice que "puntero apunta a edad". Ahora desde el puntero se puede obtener el valor almacenado en edad usando el asterisco. */
/* Ahora mostramos el valor almacenado en la dir. de memoria a la que apunta el puntero (usando *)*/
  printf("El valor almacenado en la variable es: %d", *puntero);
// Aparecerá en pantalla: El valor almacenado en la variable es: 45
 /* Ahora modificamos el valor almacenado en la variable a través del puntero que apunta a ella usando *  */
  *puntero=7;
   printf("El valor almacenado en la variable es: %d",edad);
// Aparecerá en pantalla: El valor almacenado en la variable es: 7
   return 0;
}
```

¿Para qué usar punteros?

Supongamos que queremos intercambiar el valor de dos variables en una función y mostrarlas en el programa principal (main).

```
void intercambio a,( int a,int b)
  {     int    t;
                t = a;
                a = b;
                b = t;
  }
 int  main ()
  {
                int    c,d;
                c = 54;
                d = 75;
                intercambio  (c,d);

      return 0 ;
  }
```

Veamos que pasa en la memoria de nuestro ordenador.

- Función main()
- Espacio para la variable c (posición de memoria x)
- Espacio para la variable d (posición de memoria y)
- Inicialización de las variables
- Intercambio (c,d)
- Fin de main()
- Función intercambio
- Código de la función intercambio
- Espacio privado para almacenar los parámetros (posición de memoria z)

En este último compartimiento almacenamos los valores de nuestros parámetros, que serán respectivamente 54 y 75.

Después de la ejecución de intercambio en esta zona de memoria, los valores están intercambiados, nuestro parámetro a se corresponde con la variable c en la llamada a intercambio contendrá el valor 75, y el parámetro b, correspondiente a d en la función main, contendrá el valor 54. Esto es lo que se encuentra almacenado en la zona privada de memoria de la función.

Con este esquema, y cuando la función intercambio termine su ejecución y se devuelva el control al programa principal main, los valores de c y d no habrán cambiaron, puesto que los compartimientos o posiciones de memoria x e y no han sido tocados por la función intercambio, que solo ha actuado sobre el compartimiento z.

Ejemplo correcto

Si declaramos ahora nuestra función intercambio como sigue:

```
{ void   intercambio (int *p1,int *p2)
         {
                  int     t;
                  t = *p1;    /*Metemos en t el contenido de p1 */
                 *p1 = *p2;  /* Contenido de p1 = contenido de p2 */
                  *p2 = t;
          return;
          }
```

La llamada en el main sería:

Intercambio(&c, &d);

Tendremos el mismo esquema de nuestra memoria que antes pero, en lugar de almacenar en la zona privada de la función intercambio para los parámetros los valores 54 y 75, tenemos almacenados en ella los compartimientos en los que se encuentran, esto es, hemos almacenado las posiciones x e y en lugar de 54 y 75. De esta forma, accedemos mediante un puntero a las variables c y d del programa principal que se encuentran en las

posiciones x e y, modificándolas directamente de manera que, al regresar al programa principal, sus valores se encuentran ahora intercambiados.