UNIVERSIDAD DE EL SALVADOR
FACULTAD DE INGENIERIA Y ARQUITECTURA
ESCUELA DE INGENIERIA DE SISTEMAS INFORMATICOS

Punteros

Los punteros son una de las características más útiles y a la vez más peligrosas de que dispone el lenguaje C. En dicho lenguaje, se permite declarar una variable que contiene la dirección de otra variable, o sea, un puntero. Cuando se declara un puntero éste contiene una dirección arbitraria, si leemos a dónde apunta nos dará un valor indefinido y si se escribe en tal dirección estamos variando el contenido de una posición de memoria que no conocemos por lo que podemos hacer que el sistema tenga comportamientos no deseados.

Antes de hacer uso de un puntero debemos asignarle una dirección de memoria en nuestro espacio de trabajo.

Uso.

- Es el medio por el cual, las funciones modifican sus argumentos.
- Se utilizan para asignación dinámica de memoria
- Mejor uso de memoria.

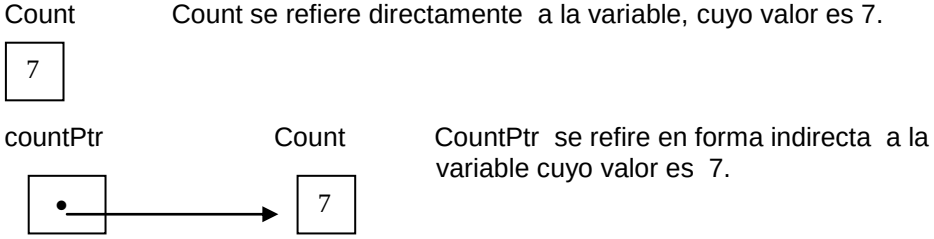
Definición:

Los apuntadores son variables que contienen direcciones de memoria como sus valores. Por lo regular una variable contiene directamente un valor específico. Un apuntador, por otra parte, contiene la dirección de una variable que contiene un valor específico. En este sentido, un nombre de variable se refiere directamente a un valor y un apuntador se refiere indirectamente a un valor.

El referirse a un valor a través de un apuntador se le conoce como indirección. Los apuntadores, como cualquier otra variable, deben ser declarados antes de que puedan ser utilizados.

Las direcciones de memoria dependen de la arquitectura de la computadora y de la gestión que el sistema operativo haga de ella. En lenguaje ensamblador se debe indicar numéricamente la posición física de memoria en que queremos almacenar un dato. De ahí que este lenguaje dependa tanto de la máquina en la que se aplique. En C no debemos, ni podemos, indicar numéricamente la dirección de memoria, si no que utilizamos una etiqueta que conocemos como variable. Lo que nos interesa es almacenar un dato, y no la localización exacta de ese dato en memoria.

La declaración int *countPtr, count;



Declara la variable countPtr siendo del tipo int* (es decir, un apuntador a un valor entero) y se lee, “countPtr es un apuntador a int”, o bien “countPtr apunta a un objeto del tipo entero”. También, la variable count se declara como un entero, no un apuntador a un entero. El * sólo se aplica a countPtr en la declaración. Cuando el * se utiliza de esta forma en una declaración,

indica que la variable que se está declarando es un apuntador. Los apuntadores pueden ser declarados para apuntar a objetos de cualquier tipo de datos.

Los apuntadores pueden ser inicializados cuando son declarados o en un enunciado de asignación. Un apuntador puede ser inicializado a 0, NULL, o a una dirección. Un apuntador con el valor NULL apunta a nada. NULL es una constante simbólica, definida en el archivo de cabecera <stdio.h>. Inicializar un apuntador a cero es equivalente a inicializar un apuntador a NULL, pero es preferible NULL. Cuando se asigna 0, primero se convierte a un apuntador del tipo apropiado. El valor 0 es el único valor entero que puede ser directamente asignado a una variable de apuntador.

Hay tantos tipos de punteros como tipos de datos, aunque también pueden declararse punteros a estructuras más complejas (funciones, struct, ficheros...) e incluso punteros vacíos (void) y punteros nulos (NULL).

El & u operador de dirección, es un operador unario que regresa la dirección de su operando.

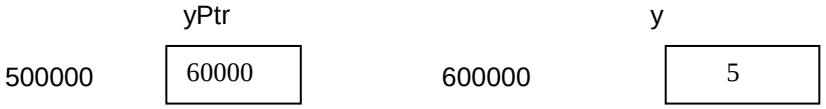
Por ejemplo, suponiendo las declaraciones

```
Int y  =  5;
Int *yPtr;
```

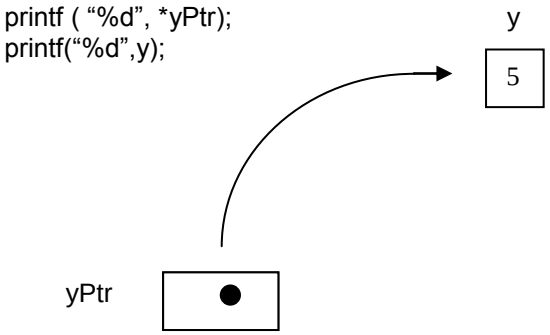
El enunciado yPtr = &y;

Asigna la dirección de la variable y a la variable de apuntador yPtr. La variable yPtr se dice entonces que "apunta a" y.

La figura posterior muestra una representación esquemática de la memoria, después de que se ejecuta la asignación anterior y la representación del apuntador en memoria, suponiendo que la variable entera y se almacena en la posición 600000, y la variable apuntador yPtr se almacena en la posición 50000. El operando de operador de dirección debe ser una variable; el operador de dirección no puede ser aplicando a constantes, a expresiones, o a variables declaradas con la clase de almacenamiento register.



Representación en memoria de **y** y yPtr .



Utilizar a * d esta forma se conoce como desreferenciar a un apuntador. La especificación de conversión de printf %p extrae la localización de memoria en forma de entero hexadecimal.

Note que la dirección de **y** y el valor de **yPtr** son idénticos en la salida, confirmando así que de hecho la dirección de **y** ha sido asignada a la variable de apuntador **yPtr**. Los operadores & y * son complementos el uno del otro –cuando se aplican ambos de manera consecutiva a aPtr, en cualquier orden, el mismo resultado será impreso.

Sintaxis: Tipo * identificador;

Ejemplo: char *text;
Int num, *conta;

Al trabajar con punteros se emplean dos operadores específicos:

- Operador de dirección: **&** Representa la dirección de memoria de la variable que le sigue:

  &fnum representa la dirección de fnum.

- Operador de contenido o indirección: *****

El operador *, conocido comúnmente como el operador de indirección o de desreferencia, regresa el valor del objeto hacia el cual su operando apunta (es decir, un apuntador). Por ejemplo, el enunciado

```
float  altura = 26.92,  *apunta;
apunta = &altura; //inicialización del puntero
printf("\n%f", altura);      //salida 26.92
printf("\n%f", *apunta);
```

No se debe confundir el operador * en la declaración del puntero:

```
int *p;
```

Con el operador * en las instrucciones:

```
*p = 27;
printf("\nContenido = %d", *p);
```

Ejemplo 1:

Num = 5
Conta = # /* Asigna la dirección de num */
Destino = conta; **/*** Asigna el valor de num a destino */

```
void fn() {
  int i;
  int *PI;
  PI = &i; /* PI ahora apunta a i */
  PI = 10; /*  Asignamos 10 a i */
}
```

Variable	Contenido	Dirección
		100
I	10	102
		104
PI	102	106
		108

Las variables tipo puntero son normalmente declaradas como cualquier otra variable excepto por la adición del carácter binario unario (*) en una expresión el unario & significa **la dirección de**. Y el unario (*) significa **la dirección que apunta o contenido de.**

Entonces se pudiese ver la primera instrucción como:

"asigne la dirección de I en PI".

La segunda asignación es:

"asigne 10 en la dirección que apunta PI".

Ejemplo 2:

```
void main(void) {
    int a, b, c, *p1, *p2;
    void *p;
    p1 = &a;            // Paso 1. La dirección de a es asignada a p1
    *p1 = 1;            // Paso 2. p1 (a) es igual a 1. Equivale a a = 1;
    p2 = &b;            // Paso 3. La dirección de b es asignada a p2
    *p2 = 2;            // Paso 4. p2 (b) es igual a 2. Equivale a b = 2;
    p1 = p2;            // Paso 5. El valor del p1 = p2
    *p1 = 0;            // Paso 6. b = 0
    p2 = &c;            // Paso 7. La dirección de c es asignada a p2
    *p2 = 3;            // Paso 8. c = 3
    printf("%d %d %d\n", a, b, c);      // Paso 9. ¿Qué se imprime?
    p = &p1;            // Paso 10. p contiene la dirección de p1
    *p = p2;            // Paso 11. p1= p2;
    *p1 = 1;            // Paso 12. c = 1
    printf("%d %d %d\n", a, b, c);      // Paso 13. ¿Qué se imprime?
  }
```

Inicialización de punteros.

Se tiene el siguiente formato para declarar un puntero:

< Almacenamiento > < Tipo > * < Nombre > = < Expresión >

Del cual:

Si <Almacenamiento> es extern o static, <Expresion> deberá ser una expresión constante del tipo <Tipo> expresado.

Si <Almacenamiento> es auto, entonces <Expresion> puede ser cualquier expresión del <Tipo> especificado.

Ejemplos:

1. La constante entera 0, NULL (cero) proporciona un puntero nulo a cualquier tipo de dato:
   ```
   int *p;
   p = NULL; //actualización
   ```

2. El nombre de un array de almacenamiento static o extern se transforma según la expresión:
   a)
   ```
   float mat[12];
   float *punt = mat;
   ```
   b)
   ```
   float mat[12];
   float *punt = &mat[0];
   ```

3. Un “cast” puntero a puntero:
   ```
   int *punt = (int *) 123.456;
   ```
   Inicializa el puntero con el entero.

4. Un puntero a carácter puede inicializarse en la forma:
   ```
   char *cadena = “Esto es una cadena”;
   ```

5. Equivalencia: Dos tipos definidos como punteros a objeto P y puntero a objeto Q son equivalentes sólo si P y Q son del mismo tipo.
   Aplicado a matrices:
   nombre_puntero = nombre_matriz;

6. Asignarle memoria dinámica a través de una función de asignación de memoria. Las funciones más habituales son calloc y malloc, definidas en el fichero alloc.h o bien en stdlib.h
   void *malloc(size_t size)
   void *calloc(size_t n_items, size)
   Una inicialización de un puntero a un tipo T tendría la forma:
   p = (T*)malloc(sizeof(T));

Punteros y arreglos unidimensionales

El nombre de un array es realmente un puntero al primer elemento de ese array.

Sin embargo, si **mat** es un array unidimensional, la dirección del primer elemento puede ser expresada tanto como **&mat[0]** o simplemente como **mat.** La dirección del elemento (i+1) se puede expresar como &mat[i] o como (mat+i). En la expresión (mat+i) **mat** representa una dirección e i un número entero. Además **mat** es el nombre de un array cuyos elementos pueden ser caracteres, enteros, números en coma flotante, etc. Por tanto, no estamos simplemente añadiendo valores numéricos. Más bien estamos especificando una localización que está i elementos del array detrás del primero, con lo cual la expresión (mat+i) es una representación simbólica de una dirección en vez de una expresión aritmética.

Si **&mat[i]** y **(mat+i)** representan la **dirección** del i-ésimo elemento de mat, **mat[i]** y ***(mat+i)** representan el **contenido** de esa dirección: el valor del i-ésimo elemento de **mat**.

Sea el array de una dimensión:
int **mat**[ ] = {2, 16, -4, 29, 234, 12, 0, 3};
en el que cada elemento, por ser tipo int, ocupa dos bytes de memoria.
Suponemos que la dirección de memoria del primer elemento, es 1500:
&mat[0] es 1500
&mat[1] será 1502
&mat[7] será 1514

En total los 8 elementos ocupan 16 bytes.
Podemos representar las direcciones de memoria que ocupan los elementos del array, los datos que contiene y las posiciones del array en la forma:

Dirección	**1500**	**1502**	**1504**	**1506**	**1508**	**1510**	**1512**	**1514**
	2	**16**	**-4**	**29**	**243**	**12**	**0**	**3**
Elemento	**mat[0]**	**mat[1]**	**mat[2]**	**mat[3]**	**mat[4]**	**mat[5]**	**mat[6]**	**mat[7]**

APLICACIÓN:

```
#include <stdio.h>
#include <conio.h>
int mat[5]={2, 16, -4, 29, 234, 12, 0, 3}, i;          //declaradas como globales
void main()  {
    printf("\n%d", &mat[0]);       //resultado: 1500 (dirección de mem)
    printf("\n%p", mat);           //resultado: 1500 ( "  "  "  "  "  )
    i++;                           //i=1
    printf("\n%p", mat+i);         //resultado: 1502 ( "  "  "  "  "  )
    printf("\n%d", *(mat+i));      //resultado: 16 (valor de mat[1] o valor
    getch();                    //en la dirección 1502
}
```

Deducimos que podemos acceder a los elementos del array de dos formas:
- mediante el subíndice.
- mediante su dirección de memoria.

Operaciones con punteros

Un valor entero se puede sumar al nombre de un array para acceder a uno de sus elementos. El valor entero es interpretado como el índice del array; representa la localización relativa del elemento deseado con respecto al primero. Esto funciona porque todos los elementos del array son del mismo tipo y por tanto cada elemento ocupa un mismo número de celdas de memoria.

El número de celdas que separan a dos elementos del array dependerá del tipo de datos del array, pero de esto se encarga el compilador automáticamente. Este concepto se puede extender a variables puntero. En particular, un valor entero puede ser sumado o restado a una variable puntero, pero el resultado de la operación debe ser interpretado con cuidado. Supongamos que es una variable puntero que representa la dirección de una variable x. Se pueden escribir expresiones como ++px, --px, (px+3). Cada expresión representa una dirección localizada a cierta distancia de la posición original representada por px. La distancia exacta será el producto de la cantidad entera por el número de bytes que ocupa cada elemento al que apunta px.

Resumen de operaciones que se pueden realizar sobre punteros:

1. A una variable puntero se le puede asignar la dirección de una variable ordinaria (pv=&v).
2. A una variable puntero se le puede asignar el valor de otra variable puntero, siempre que ambas apunten al mismo tipo de dato.
3. A una variable puntero se le puede asignar el valor nulo (NULL).
4. Una cantidad entera puede ser sumada o restada a una variable puntero.
5. Una variable puntero puede ser restada de otra con tal de que ambas apunten a elementos del mismo array.
6. Dos variables puntero pueden ser comparadas siempre que ambas apunten a datos del mismo tipo.
7. No se permiten operaciones aritméticas con punteros. Así una variable puntero no puede ser multiplicada por una constante, dos punteros no pueden sumarse, etc.

Operaciones sobre punteros.

```
While (*ps)
{
  if (islower (*ps)){
     *ps = toupper (*ps);
  }
  ps++;
  }
```

Situación del puntero.	*Dato*	*Dirección*
Ps Antes	E	100
Ps después	S	102
	T	103
	R	104
	U	105
	C	106

	T	107
	U	108
	R	109
	A	10 A
	\0	

EJEMPLO:

```
# include <ctype.h>
void uppercase(char *ps){
        while(*ps){
                if (islower (*ps)){
                        ps = toupper(ps);
                }
                ps++;
        }
}

void fn(){
        char *pString;
        pString = “Estructura”;
        uppercase(pString);
}
```

‘E’	‘S’	‘T’	‘R’	‘U’	‘C’	‘T’	‘U’	‘R’	‘A’	\0