LENGUAJE C

Tema 6 – Punteros

www.profmatiasgarcia.com.ar

QUÉ ES UN PUNTERO?

- Un puntero es una variable que contiene la dirección de memoria de un dato o de otra variable que contiene al dato. Esto quiere decir que el puntero apunta al espacio físico donde está el dato o la variable.
- Las variables de tipo puntero representan direcciones donde almacenar valores. Es importante diferenciar entre puntero (espacio de memoria donde se almacena la dirección) y la propia dirección apuntada por el puntero (su valor).
- La forma general es:

```
tipo *nombre_var;
```

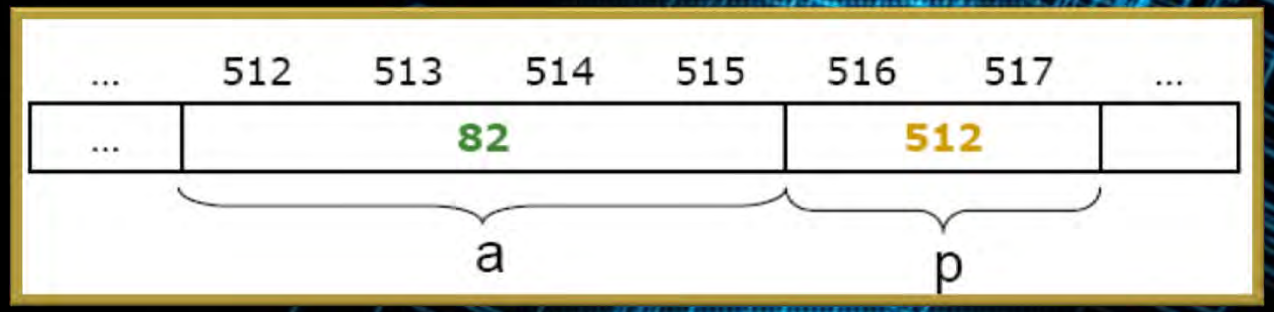
- * retorna el valor de la variable localizada en la dirección especificada por el operando.
- Ejemplos:

```
int *a; //puntero a un entero
float *a;//puntero a un flotante
int *px, y;  /* px es un puntero a entero e y es un entero */
```

PUNTEROS

- Inicialmente las variables de tipo puntero, apuntan a direcciones de memoria indeterminadas, por tanto deben ser debidamente inicializados antes de ser usadas.
- Asignación

```
int a;
int *p;
a = 82;
p = &a;
printf("%p %p %p %d", &a, p, &p, *p);
```



Dirección de a

Contenido de p

Dirección de p

Lo apuntado por p

- Los punteros admite dos operadores básicos:
  - Si `px` es un puntero (dirección): `*px` es el contenido del puntero (el valor almacenado en la dirección). Lo apuntado por el puntero.
  - Si `x` es una variable: `&x` es la dirección de memoria donde está almacenada la variable.

Punteros 2

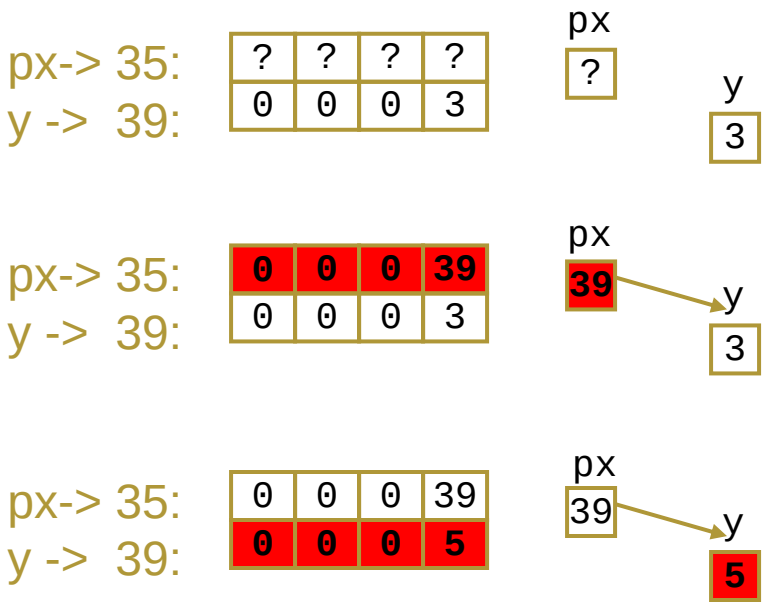
Código

```
int main()
{
   int *px, y=3;

   px = &y;
   /* px apunta a y*/

   *px = 5;
   /* y vale 5 */
}
```

Dirección Contenido Gráfica



Aritmética de punteros

- Las operaciones soportadas sobre punteros son:
  - Suma y resta de valores enteros (+,-,++ y –).
  - Comparación y relación (<,>,<=,>=,== y !=).
  - Valor booleano (comparación con NULL).
- El puntero “avanza” según el tamaño de lo apuntado

```
p++; --p; // una posición si apuntan a char
p++; --p; // dos posiciones si apuntan a int
p++; --p; // cuatro posiciones si apuntan a long int, etc.
```

- Acceso a la i-ésima posición del vector

```
*(p+i)
p[i]
```

- Los punteros se pueden restar (resultado: desplazamiento)
- Los punteros no se suman !
- Un puntero se inicializa como cualquier otra variable, aunque los únicos valores significativos son NULL (puntero nulo) o la dirección de un objeto previamente definido.

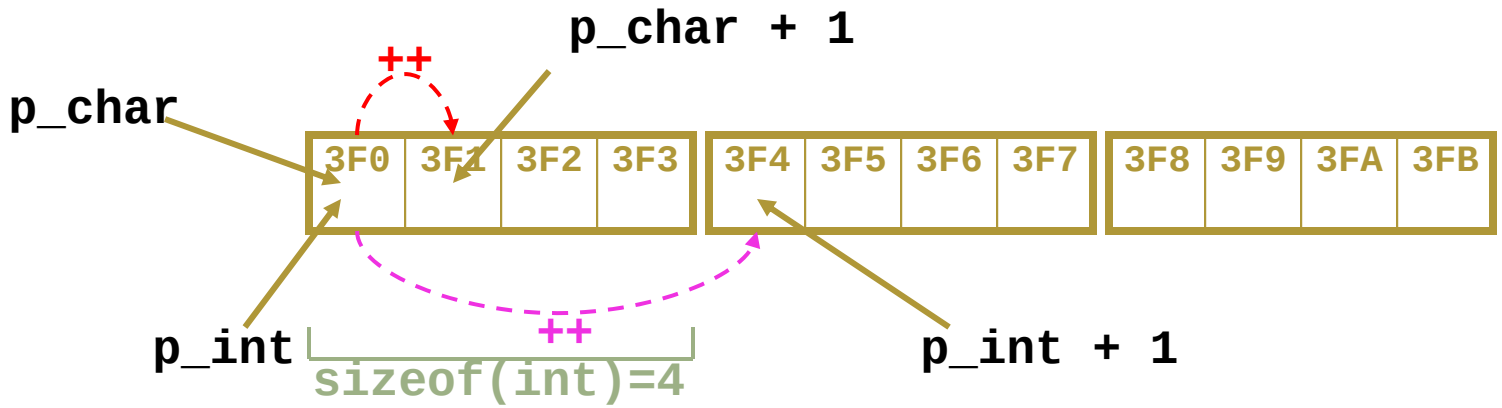
```
void copiar(char* dest,
    const char* orig)
{
    if(orig && dest)
       while(*orig)

        *dest++=*orig++;
}
```

Aritmética de punteros 2

Las operaciones de suma o resta sobre punteros modifican el valor del dependiendo del tipo del puntero:

```
Int *p_int;
char *p_char;
p_int = p_char;
p_int++;  /* Suma sizeof(int) */
p_char++; /* Suma sizeof(char) */
```



PUNTEROS CONSTANTES

- Una declaración de un puntero precedida por const, hace que el objeto apuntado sea tratado como constante, no sucediendo lo mismo con el puntero.

```
Int a = 10, b = 20;
const int *p = &a; //objeto constante y p variable
*p = 15; // error: el objeto apuntado por p es constante
p = &b; // correcto: p pasa a apuntar a otro objeto
```

- Si lo que se pretende es declarar un puntero constante, procederemos así:

```
Int a = 10, b = 20;
Int * const p = &a; //objeto variable y p constante
*p = 15; // correcto: el objeto apuntado por p es variable
P = &b; // error: p es constante
```

PUNTEROS Y VECTORES

El identificador de una variable array tiene el valor de la dirección de comienzo del mismo. Por lo tanto, su valor puede usarse como un puntero.

```
int *pb,*pc;
int a[5]={10,20,30,40,50};
```


```
pb=a;
*pb=11;
```


```
pc=&a[3];
*pc=44;
```

PUNTEROS Y VECTORES 2

Los arrays de varias dimensiones sí se diferencian de los punteros múltiples:

Matriz de 2 dimensiones:

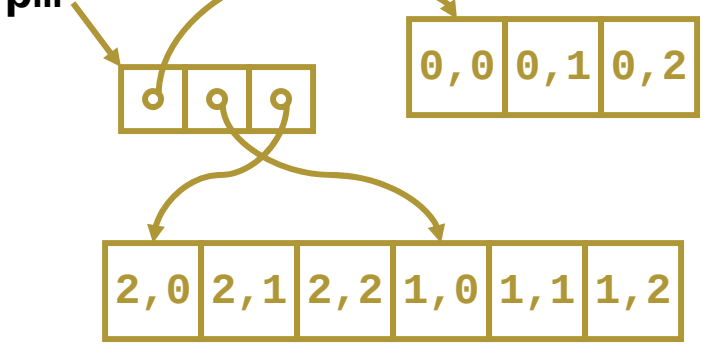
```
int mx[3][3];
```

mx


```
Pm = mx; /* ERROR */
pm[0]= mx[1]; /* OK */
Pm[0][1]= mx[1][2] /* OK */
```

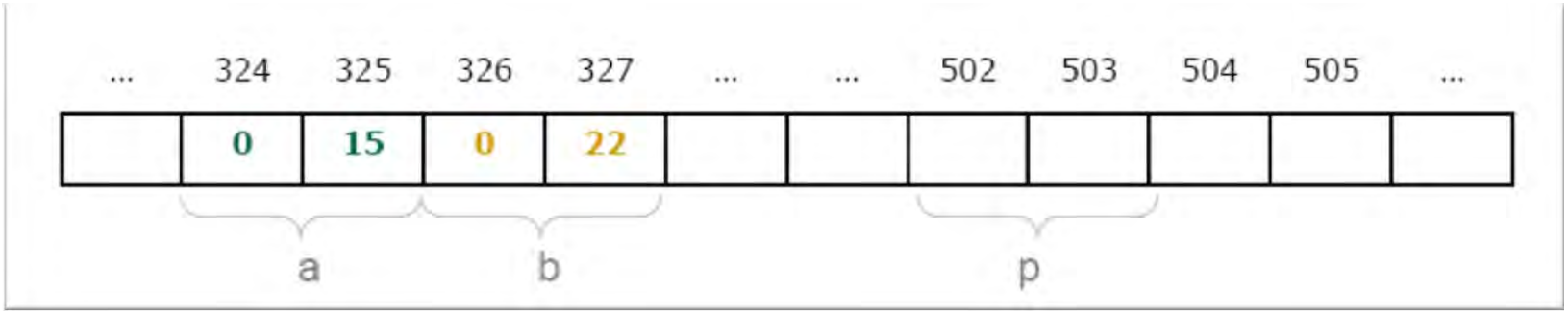
Puntero múltiple:

```
int **pm;
```



Punteros como parámetros

```
void f(int * p)
{
  *p = 5;
}
void g(void)
{
  int a = 15, b = 22;
  f(&a);
  f(&b);
  printf("%d %d", a, b);
}
```



Pasar un vector a una función

- Cuando una matriz es un argumento a una función, solo se pasa la dirección del primer elemento de la matriz, no una copia de la matriz entera.
- En C el nombre del vector sin índice es un puntero al primer elemento.
- Existen tres maneras de declarar un parámetro que sea recibido como un puntero vector:
  - Vector del mismo tipo y tamaño como se usa para llamar a la función
  - un puntero
  - Vector sin tamaño

```
void display(int n[10]);
int main()
{
  int t[10],i;
  for(i=0;i<10;++i)t[i]=i;
  display(t);//pasar vector t a
    función
  return 0;
}
void display(int n[10])
{
  int i;
  for(i=0;i<10;i++)
    printf("%d",n[i]);
}
```

Pasar un vector a una función

```
void display(int *n);
int main()
{
  int t[10],i;
  for(i=0;i<10;++i)t[i]=i;
  display(t);// pasar vector t a
    función
  return 0;
}

void display(int *n)
{
  int i;
  for(i=0;i<10;i++)
   printf("%d",n[i]);
}
```

```
void display(int n[]);
int main()
{
  int t[10],i;
  for(i=0;i<10;++i)t[i]=i;
  display(t);//pasar vector t a
    función
  return 0;
}

void display(int n[])
{
  int i;
  for(i=0;i<10;i++)
    printf("%d",n[i]);
}
```

Relación entre arrays y punteros

- 1) El nombre de un array es una dirección

```
int v[10];
v → &v[0]
```

- Consecuencia 1

```
int edad; char nombre[32];
scanf("%d %s", &edad, nombre);
scanf("%d %s", &edad, &nombre[0]);
```

- Consecuencia 2

```
void inicia(int v[], int len)
{
int i;
for (i=0; i<len; i++)
v[i] = 0;
}
int main() {
int v[10];
inicia(v, 10);
inicia(&v[0], 10);
return 0;}
```

- Puede ponerse

```
void inicia(int *v, int len)
{ …
}
```

Relación entre arrays y punteros

- 2) Los arrays pueden usar notación de punteros

```
int v[10];
```

- Para acceder a los elementos

```
v[0] v[1] v[2] v[3] …
*v *(v+1) *(v+2) *(v+3) …
```

- Generan el mismo código
- 3) Los punteros pueden usar notación de arrays

```
int v[10];
int *p = v;
```

- Para acceder a los elementos

```
*p *(p+1) *(p+2) *(p+3) …
p[0] p[1] p[2] p[3] …
```

Relación entre matrices y punteros

- Un array multidimensional es, en realidad, una colección de vectores. Según esto, podemos definir un array bidimensional como un puntero a un grupo contiguo de arrays unidimensionales. Las declaraciones siguientes son equivalentes:

```
int dat[fil][col]  ⟶  int (*dat)[col]
```

- En general:

```
tipo_dato nombre[dim1][dim2]. . . . .[dimp]; equivale a:
tipo_dato (*nombre)[dim2][dim3]. . . . .[dimp];
```

Relación entre matrices y punteros

- Si tengo la matriz x[10][20] y quiero acceder al elemento de la fila 3 y la columna 6, lo hago escribiendo x[2][5]. Con notación de punteros, lo que hacemos es considerar que es un array formado por 10 vectores de 20 elementos cada uno:

```
*(*( x+2 ) +5)
```

- ya que x+2 es un puntero a la fila 3. Por tanto el contenido de dicho puntero, *(x+2), es la fila 3. Si me desplazo 5 posiciones en esa fila llego a la posición *(x+2)+5, cuyo contenido es *(*(x+2)+5). Las siguientes expresiones con punteros son válidas:

```
**x  ------> x[0][0] ;
*(*x+1) ---> x[0][1];
*(*(x+1)) --> x[1][0]
**(x+1) ----> x[1][0]
```

Punteros a cadenas de caracteres

- Puesto que una cadena de caracteres es una matriz de caracteres, es correcto pensar que se pueden utilizar punteros para su tratamiento.

```
Char * cadena;
```

- La dirección de memoria donde comienza una cadena viene dada por el nombre de la matriz que la contiene, y el final por el carácter \0 con el que C finaliza.

```
Char *nombre = “Matias Garcia”;
Printf (“%s”, nombre);
```

- La modificación de una matriz de caracteres por punteros

```
Char nombre[] = “Matias Garcia”;
Char *pnombre = nombre;
Pnombre[7] = “-”;
```

Estructuras y punteros

- A una variable de tipo estructura se puede acceder mediante un puntero de la misma forma que cualquier otra variable.
- La forma de hacerlo es:

```
nombre_de_variable_puntero -> miembro_de_la_estructura
```

Operador que permite seleccionar un miembro de estructura

- Ejemplo

```
Struct Tusuario persona1, *persona2;
//leo los datos de persona1
Persona2 = &persona1;
printf("%s", persona2->nombre);
….
```

Asigna al puntero persona2 la dirección de persona1

Accedo a persona1 a través del puntero persona2

Punteros a punteros

- Un puntero puede almacenar una dirección de otro puntero.
- El valor final apuntado puede obtenerse en forma directa.

```
int **puntero; //puntero a puntero a un objeto int.
int n, *j, **p;  n=4562;  j=&n;  p=&j;
```

- El tipo de objeto apuntado después de una doble indirección puede ser de cualquier clase.
- Permite manejar arrays de múltiples dimensiones con notaciones del tipo ***mat, de múltiple indirección que pueden generar problemas si el tratamiento no es el adecuado.

Punteros a funciones

- Un puntero a función es una variable cuyos posibles valores son direcciones en las que se encuentran funciones.
- La dirección de una función se obtiene con el nombre de la función sin paréntesis ni argumentos (no hace falta el operador &): `pf = nombre_función;`
- Si el puntero no apunta a ninguna función se inicializa a NULL: `pf = NULL;`
- Se puede realizar de dos maneras la invocación de funciones mediante punteros :

```
(*pf)(lista_parametros_actuales);
pf (lista_parametros_actuales);
```

- Los punteros a funciones permiten pasar funciones como parámetros en la llamada a otras funciones

Expresiones con punteros

`int *p;`	p es un puntero a un entero
`int *p[10];`	p es un array de 10 punteros a enteros
`int (*p)[10];`	p es un puntero a un array de 10 enteros
`int *p(void);`	p es una función que devuelve un puntero a entero
`int p(char *a);`	p es una función que acepta un argumento que es un puntero a carácter, devuelve un entero
`int *p(char *a);`	p es una función que acepta un argumento que es un puntero a carácter, devuelve un puntero a entero
`int (*p)(char *a);`	p es un puntero a función que acepta un argumento que es un puntero a carácter, devuelve un puntero a entero
`int (*p(char *a))[10];`	p es una función que acepta un argumento que es un puntero a carácter, devuelve un puntero a un array de 10 enteros
`int p(char (*a)[]);`	p es un puntero a función que acepta un argumento que es un puntero a un array de caracteres, devuelve un puntero a entero
`int p(char *a[]);`	p es un puntero a función que acepta un argumento que es un array de punteros a caracteres, devuelve un puntero a entero
`int *p(char a[]);`	p es una función que acepta un argumento que es un array de caracteres, devuelve un puntero a entero
`int *p(char (*a)[]);`	p es una función que acepta un argumento que es un puntero a un array de caracteres, devuelve un puntero a entero

Expresiones con punteros

`int *p(char *a[]);`	p es una función que acepta un argumento que es un puntero a un array de punteros a caracteres, devuelve un puntero a entero
`int (*p)(char (*a)[]);`	p es una función que acepta un argumento que es un puntero a un array de caracteres, devuelve un puntero a entero
`int *(*p)(char (*a)[]);`	p es un puntero a una función que acepta un argumento que es un puntero a un array de punteros a caracteres, devuelve un puntero a entero
`int *(*p)(char *a[]);`	p es un puntero a una función que acepta un argumento que es un array de punteros a caracteres, devuelve un puntero a entero
`int (*p[10])(void);`	p es una array de 10 punteros a función, cada función devuelve un entero
`int (*p[10])(char * a);`	p es una array de 10 punteros a función; cada función acepta un argumento que es un puntero a carácter y devuelve un entero.
`int *(*p[10])(char a);`	p es una array de 10 punteros a función; cada función acepta un argumento que es un carácter, y devuelve un puntero a entero.
`char *(*p[10])(char * a);`	p es una array de 10 punteros a función; cada función acepta un argumento que es un carácter, y devuelve un puntero a caracter.

WEBGRAFÍA & LICENCIA:

- Textos tomados, corregidos y modificados de diferentes páginas de Internet, tutoriales y documentos, entre los que destaco el libro: *C/C++ Curso de programación*, 2da Ed, Javier Ceballos, Alfaomega Ra-Ma.
- Este documento se encuentra bajo Licencia Creative Commons 2.5 Argentina (BY-NC-SA), por la cual se permite su exhibición, distribución, copia y posibilita hacer obras derivadas a partir de la misma, siempre y cuando se cite la autoría del Prof. Matías E. García y sólo podrá distribuir la obra derivada resultante bajo una licencia idéntica a ésta.
- Autor:

Matías E. García

Prof. & Tec. en Informática Aplicada
www.profmatiasgarcia.com.ar
info@profmatiasgarcia.com.ar