Punteros.

Programación I – Laboratorio I.

Tecnicatura Superior en Programación.

UTN-FRA

Autor: *Lic. Mauricio Dávila. Basado en el apunte "El lenguaje de programación C" de la* Universidad de Valencia

Revisores: *Ing. Ernesto Gigliotti*

Versión : 1

Índice de contenido

1 Punteros

Los punteros son una de las poderosas herramientas que ofrece el lenguaje C a los programadores, sin embargo, son también una de las más peligrosas, el uso de punteros sin inicializar, etc., es una fuente frecuente de errores en los programas de C, y además, suele producir fallas muy difíciles de localizar y depurar.

1.1 Definición

Un puntero es una variable que contiene una dirección de memoria, normalmente esa dirección es una posición de memoria de otra variable, por lo cual se suele decir que el puntero "apunta" a la otra variable.

1.2 Declaración de un puntero

La sintaxis de la declaración de una variable puntero es:

```
tipo* nombre;
```

El tipo base de la declaración sirve para conocer el tipo de datos al que pertenece la variable a la cual apunta la variable de tipo puntero. Esto es fundamental para poder leer el valor que almacena la zona de memoria apuntada por la variable puntero y para poder realizar ciertas operaciones aritméticas sobre los mismos.

Algunos ejemplos de declaración de variables puntero son:

```
int* a;

char* p;

float* f;
```

1.3 Operadores

Existen dos operadores especiales de los punteros, el operador '&' encargado de retornar la dirección de memoria de una variable y el operador '*' encargado de retornar el valor de la variable apuntada por un puntero.

Si declaramos:

```
int* a;

int b;
```

Y hacemos:

```
a = &b;
```

La variable puntero '**a**' contendrá la dirección de memoria de la variable '**b**'.

Veámoslo otro ejemplo:

```
int* a;
int b,c;
```

Y hacemos:

```
b=15; // Asignamos el valor 15 a la variable 'b'
a=&b; // Obtenemos la posición de memoria de 'b' con el  operador '&'
c=*a; // Copiamos el contenido apuntado por el puntero 'a'
```

Entonces la variable '**c**' contendrá el valor 15, pues '***a**' devuelve el contenido (o valor) de la dirección a la que "apunta" la variable puntero, y con anterioridad hemos hecho que '**a**' contenga la dirección de memoria de la variable '**b**' usando para ello el operador '**&**'.

1.4 Asignación de punteros

Es posible asignar el valor de una variable de tipo puntero a otra variable de tipo puntero.

Por ejemplo:

```
int* a;
int* b;
int  c;
a=&c; // Obtenemos la posición de memoria de 'c' con el  operador '&'
b=a;  // Asignamos el valor del puntero 'a' al puntero 'b'
```

Entonces b contiene el valor de a, y por ello, b también "apunta" a la variable c.

1.5 Comparación de punteros

Sobre las variables de tipo puntero es posible realizar operaciones de comparación, veamos un ejemplo:

```
int* punteroA;
int* punteroB;
int  auxiliarC , auxiliarD;

punteroA = &auxiliarC; // Obtenemos la posición de memoria de 'auxiliarC'
punteroB = &auxiliarD; // Obtenemos la posición de memoria de 'auxiliarD'

if (punteroA<punteroB)
      printf("El punteroA  apunta a una dirección más baja que punteroB");
else if (punteroA>punteroB)
      printf("El punteroB  apunta a una dirección más baja que punteroA");
```

1.6 Aritmética de punteros

Sobre las variables de tipo puntero es posible utilizar los operadores +, -, ++ y --. Estos operadores incrementan o decrementan la posición de memoria a la que "apunta" la variable puntero. El incremento o decremento se realiza de acuerdo al tipo base de la variable de tipo puntero, de ahí la importancia del tipo del que se declara la variable puntero.

Veamos esto con la siguiente tabla:

	Posición Actual	a++	a--	a=a+2	a=a-3
`char *a`	0xA080	0xA081	0xA07F	0xA082	0xA07D
`int *a`	0xB080	0xB084	0xB07C	0xB088	0xB06E
`float *a`	0xC080	0xC084	0xA07C	0xA088	0xA06E

Como podemos observar en la tabla cada operación sobre el puntero se rige por su tipo, por lo tanto, si tenemos:

```
tipo *a;
```

Y hacemos:

```
a = a + num;
```

La posición a la que apunta a se incrementa en:

```
nueva dirección que contiene "a" = dirección que contiene "a" + (num * sizeof(tipo))
```

Para la resta se decrementa de igual forma en:

```
nueva dirección que contiene "a" = dirección que contiene "a" - (num * sizeof(tipo))
```

Los operadores ++ y -- son equivalentes a realizar num=1, y con ello quedan obviamente explicados.

1.7 Vectores y punteros

Existe una estrecha relación entre los punteros y los vectores. Consideremos el siguiente fragmento de código:

```
char cadena[80];
char *p;
p=&cadena[0]; // equivalente a: p=cadena
```

Este fragmento de código pone en la variable puntero '**p**' la dirección del primer elemento del array '**cadena**'.

Entonces, es posible acceder al valor de la quinta posición del array mediante: cadena[4] y *(p+4) (recuérdese que los índices de los arrays empiezan en 0).

Esta estrecha relación entre los arrays y los punteros queda más evidente si se tiene en cuenta que el **nombre del array sin índice es la dirección de comienzo del array**, y si además, se tiene en cuenta que un puntero puede indexarse como un array unidimensional, por lo cual, en el ejemplo anterior, podríamos referenciar ese elemento como p[4].

Es posible obtener la dirección de un elemento cualquiera del array de la siguiente forma:

```
int arrayInt[80];
int* p1;
int* p2;
p1 = &arrayInt[4];
p2 = &arrayInt;
```

Entonces, el puntero '**p1**' contiene la dirección del quinto elemento del arrayInt y el puntero '**p2**' contiene la dirección del primer elemento del arrayInt.

Hasta ahora hemos declarado variables puntero aisladas. Es posible, como con cualquier otro tipo de datos, definir un array de variables puntero. La declaración para un array de punteros int de tamaño 10 es:

```
int* a[10];
```

Para asignar una dirección de una variable entera, llamada '**var**', al tercer elemento del array de punteros, se escribe:

```
x[2] = &var;
```

Y para obtener el valor de var:

```
*x[2];
```

Dado además, que un puntero es también una variable, es posible definir un puntero a un puntero. Supongamos que tenemos lo siguiente:

```
int a;
int *punteroInt;
int **punteroPuntero;
punteroInt = &a; // Obtenemos la posición de memoria
punteroPuntero = &puntero; // Obtenemos la posición de memoria
```

Y entonces, ¿De qué formas podemos ahora acceder al valor de la variable '**a**'?

a (forma habitual)

*punteroInt (a través del puntero)

**punteroPuntero (a través del puntero a puntero)

Esto es debido a que '**punteroPuntero**' contiene la dirección de '**punteroInt**', que contiene la dirección de '**a**'.

Este concepto de puntero a puntero podría extenderse a puntero a puntero a puntero, etc., además, existe el concepto de puntero a una función, al cual nos referiremos más adelante.