Unidad Temática N°2

~ Punteros y Cadenas ~

"No hay nada en computación que no pueda ser roto por otro nivel de direccionamiento indirecto".

~ Rob Pike

Punteros

Al ejecutar nuestro programa, por cada variable declarada, se reserva un espacio de memoria para almacenar el contenido de la misma. A cada variable declarada se le asocian 3 atributos fundamentales que son: su nombre, su tipo y su dirección de memoria.

Cuando desde nuestro programa queremos acceder a alguna de las variables, lo hacemos a través de su nombre, pero la máquina accede directamente a través de su dirección de memoria.

Ejemplo:

```
#include <stdio.h>

int main(){
   int n = 10;
   printf("n = %d\n", n);
   printf("Dir. de memoria de n = %p\n",
&n);
   return 0;
}
```

Ejecución

```
n = 10;
Dir. de memoria de n = 0xff123098
```

Nosotros podemos hacer lo mismo mediante el uso de punteros.

Un puntero es una variable como cualquier otra que hayamos utilizado, sólo que sus valores serán direcciones de memoria de otras variables.

Declaración e Inicialización

Para utilizar un puntero, al igual que las otras variables, debemos declararlo e inicializarlo.

Un puntero se declara de la siguiente manera:

```
<tipo de dato> *<nombre>;
```

Donde *<tipo de dato>* deberá corresponder con el tipo de dato de la variable apuntada, es decir, la variable cuya dirección de memoria vamos a contener en el puntero.

Para inicializar un puntero utilizaremos la constante NULL.

Ejemplos:

```
int *ptrInt = NULL;  // puntero a entero, inicializado en NULL
float *ptrFloat;     // puntero a float, sin inicializar
```

```
char *ptrCadena;     // puntero a char, sin inicializar
```

Previamente vimos un ejemplo donde se imprimía la dirección de memoria de una variable, y para esto usamos el signo & (ampersand) delante del nombre de la variable. El signo & (ampersand) es un operador que nos devuelve la dirección de memoria de una variable. Este operador podría ser utilizado para inicializar un puntero con la dirección de memoria de una variable o bien, para asignarle la dirección de memoria de una variable a un puntero en cualquier momento.

```
// Ejemplo 1
int n = 10;
int *ptrInt = &n;  // puntero a entero, inicializado con la direción de n

// Ejemplo 2
char letra = ‘p’;
char *ptr = &letra;

// Ejemplo 3
char palabra[] = “Hola”;
char *ptrPalabra = palabra;
```

Indirección de punteros

Una vez que definimos e inicializamos una variable de tipo puntero, el próximo paso es utilizarlo, ya sea para asignar un valor o bien para leer un valor desde una dirección de memoria. Para poder realizar estas acciones vamos a utilizar el operador *.

Ejemplo

Asignación	Lectura

`int n = 0;`
`int *ptr = &n; // inicializa ptr con la dir. de n`
 	
`*ptr = 20;`
`printf(“n = %d\n”, n); // imprime 20`
	`int n = 10;`
	`int * ptr = &n;`
	
	`printf(“*ptr = %d\n”, *ptr); // imprime 10;`

Tener cuidado al realizar una asignación a un puntero, verificar que el puntero esté apuntando a alguna variable.

```
int *ptr;

*ptr = 20; // Esto es un error ya que ptr está apuntando a nada.
```

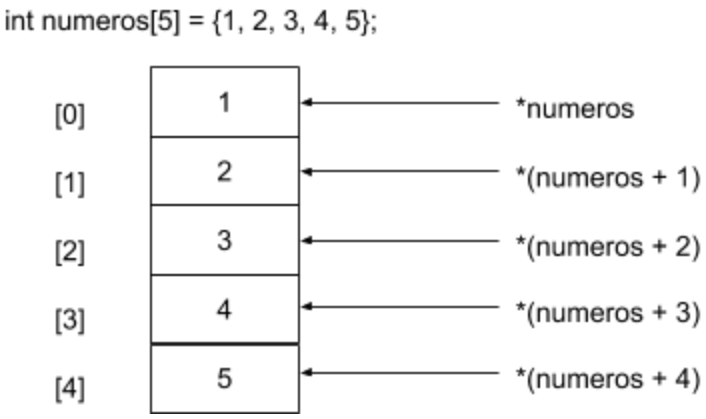
Resumiendo

Operador	Descripción
*	Definición de una variable puntero
&	Obtiene la dirección de memoria de una variable
*	Obtiene y también permite asignar el contenido de una variable puntero

Punteros y Vectores

En C, los punteros y vectores (arrays) están fuertemente relacionados. Se pueden direccionar arrays como si fueran punteros y punteros como si fueran arrays.

El nombre de un array, es un puntero a la primer posición del mismo.



Es por esta relación que para visualizar, almacenar o asignar un valor a un elemento de un array se puede utilizar notación de subíndices o *notación de punteros*.

El nombre de un array es un puntero al primer elemento del array, es decir, contiene la dirección de memoria del primer elemento del array. Se dice que un array es un puntero constante ya que **no** se puede modificar la dirección a la que apunta, sólo se puede modificar su contenido.

```
#include <stdio.h>

int main(){
	int lista[] = {1,2,3};
	printf("%d\n", *lista);
	printf("%d\n", *lista+1);
	printf("%d\n", *lista+2);

	return 0;
}
```

Del ejemplo anterior sabemos que *lista* es un array de enteros, o lo que es lo mismo decir un puntero constante a enteros. Si quisiéramos hacer que *lista* apunte a otra dirección de memoria obtendríamos un error.

```
#include <stdio.h>

int main(){
	int lista[] = {1,2,3};
	 int x = 10;
	 lista = &x; // ESTO NO SE PUEDE HACER PORQUE lista ES UN PUNTERO CONSTANTE

	return 0;
}
```

Punteros constantes y Punteros a constantes

Definición	**Puntero constante**: es un puntero que no puede cambiar la dirección de memoria a la que apunta, pero sí puede cambiar el contenido de aquello a lo que apunta.

Para definir un puntero constante, lo realizamos de la siguiente manera:

```
<tipo de dato> *const <nombre>;
```

Ejemplo

```
int x = 10;
int y = 20;
int *const p1 = &x;

*p1 = y; // esto es legal ya que cambiamos el contenido de p1.

p1 = &y; // esto no es legal ya que intenta cambiar el valor de p1.
```

Definición	**Puntero a constante**: es un puntero que puede cambiar la dirección de memoria a la que apunta, pero no puede cambiar el contenido de aquello a lo que apunta.

Para definir un puntero a constante, lo realizamos de la siguiente manera:

```
const <tipo de dato> * <nombre>;
```

Ejemplo

```
int x = 10;
int y = 20;
const int * p1 = &x;

*p1 = y; // esto no es legal ya que intenta cambiar el contenido de p1.

p1 = &y; // esto es legal ya que cambia el valor de p1 pero no su
contenido.
```

Los punteros a constantes son muy utilizados cuando definimos los parámetros de una función, para protegernos de cambiar el contenido de lo apuntado por el puntero dentro de la función.

Ejemplo:

```
#include <stdio.h>

int longitudPalabra(const char *palabra){
      int longitud = 0;
      while(*(palabra++)){
         longitud++;
      }
      return longitud;
}

int main(){
      char pal[] = "hola mundo";
      printf("%d\n", longitudPalabra(pal));
      return 0;
}
```

Aritmética de punteros

Si el puntero no es un puntero constante, el mismo puede ser modificado. Para modificar un puntero, podemos, además de asignarle la dirección de memoria de distintas variables, realizar operaciones aritméticas como sumas o restas de enteros para que el puntero apunte a *n* direcciones posteriores o anteriores.

```
#include <stdio.h>

int main(){
  char pal[] = "hola mundo";
  char *p;
  p = pal;
  p = p + 2; // p apunta a la “l” de hola
  return 0;
}
```

Recuerde	• No se puede sumar 2 punteros • No se puede multiplicar 2 punteros

	• No se puede dividir 2 punteros.

Cadenas

En C, una *cadena* o *cadena de caracteres* es un array de caracteres (char) finalizados con el caracter '\0' (contrabarra cero).

h	o	l	a	\0

Declaración e inicialización de variables de cadena

Para declarar una variable de tipo cadena, podemos utilizar una sintaxis similar a la utilizada para declarar arrays o bien, podemos utilizar una sintaxis similar a la utilizada para declarar punteros.

Ejemplo:

```
a)_ char palabra[] = "hola mundo";

b)_ char palabra2[11] = "hola mundo";

c)_ char *palabra = "hola mundo";
```

En el caso 'a', la dimensión de la cadena se asigna automáticamente a partir del contenido asignado.

En el caso 'b' lo definimos al declarar la variable. Si bien "hola mundo" posee 10 caracteres, en la declaración debemos reservar espacio para un caracter más que el caracter de finalización de la cadena, el caracter '\0'.

En el caso 'c' el sistema reserva memoria automáticamente al igual que en el caso 'a'.

¿Por qué no podemos declarar una variable de tipo cadena y luego asignarle valor?

Esto se debe a que una variable de tipo cadena es un *puntero constante* y por lo tanto no se puede cambiar la dirección de aquello a lo que apunta.

Es por este motivo que para declarar y luego asignarle valor a una variable de tipo cadena se utilizan funciones de la biblioteca string.h.

Biblioteca string.h

Nombres	Descripción
memcpy	copia n bytes entre dos áreas de memoria que no deben solaparse
memset	sobre escribe un área de memoria con un patrón de bytes dado
strcat	añade una cadena al final de otra
strncat	añade los n primeros caracteres de una cadena al final de otra
strchr	localiza un carácter en una cadena, buscando desde el principio
strrchr	localiza un carácter en una cadena, buscando desde el final
strcmp	compara dos cadenas alfabéticamente ('a'!='A')
strncmp	compara los n primeros caracteres de dos cadenas numéricamente ('a'!='A')
strcpy	copia una cadena en otra
strncpy	copia los n primeros caracteres de una cadena en otra
strlen	devuelve la longitud de una cadena
strspn	devuelve la posición del primer carácter de una cadena que no coincide con ninguno de los caracteres de otra cadena dada
strstr	busca una cadena dentro de otra
strtok	parte una cadena en una secuencia de tokens

Para más información ver http://www.cplusplus.com/reference/cstring/