Introducción a punteros

Ciencia de la Computación II

MSc. Gina Muñoz Salas

Punteros

Un puntero es una variable que almacena la dirección de memoria de otra variable.

- Los punteros impregnan el lenguaje y proporcionan gran parte de su flexibilidad.
- Ofrecen un soporte importante para la asignación de memoria dinámica.
- Están estrechamente ligados a la notación de arrays y, al usarse para apuntar a funciones, añaden otra dimensión al control de flujo en un programa.

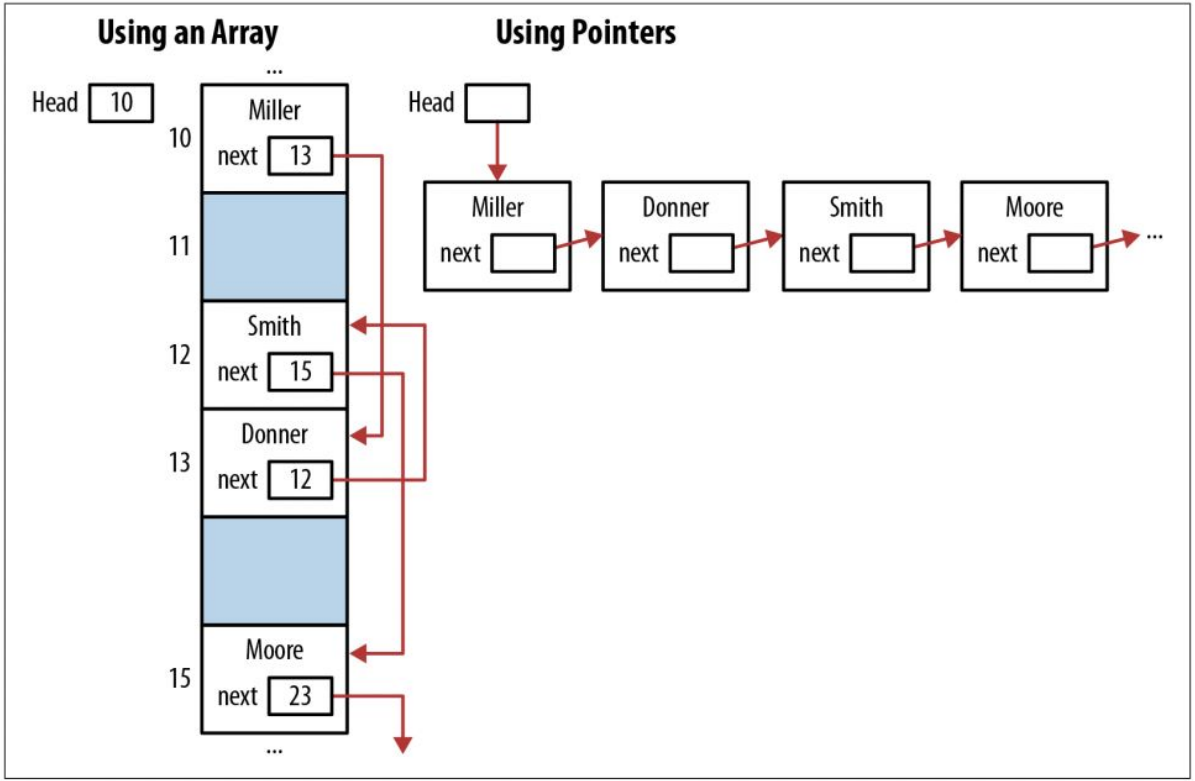
Punteros

- La clave para comprender los punteros radica en entender cómo se gestiona la **memoria** en un programa en C.
- Los punteros contienen direcciones en la memoria, por lo que comprender cómo se organiza y administra la memoria es crucial.
- Se ilustra la organización de la memoria siempre que sea útil para explicar un concepto de puntero.
- Una vez que se tiene un firme entendimiento de la memoria y las formas en que puede organizarse, comprender los punteros se vuelve mucho más fácil.

¿Por qué aprender punteros?

- Creación de código rápido y eficiente
- Proporcionar un medio conveniente para abordar diversos problemas
- Soporte para la asignación de memoria dinámica
- Hacer expresiones compactas y concisas
- Facilitar el paso de estructuras de datos por puntero sin incurrir en un gran sobrecosto
- Proteger los datos pasados como parámetro a una función

Ejemplo: Implementación de una lista enlazada

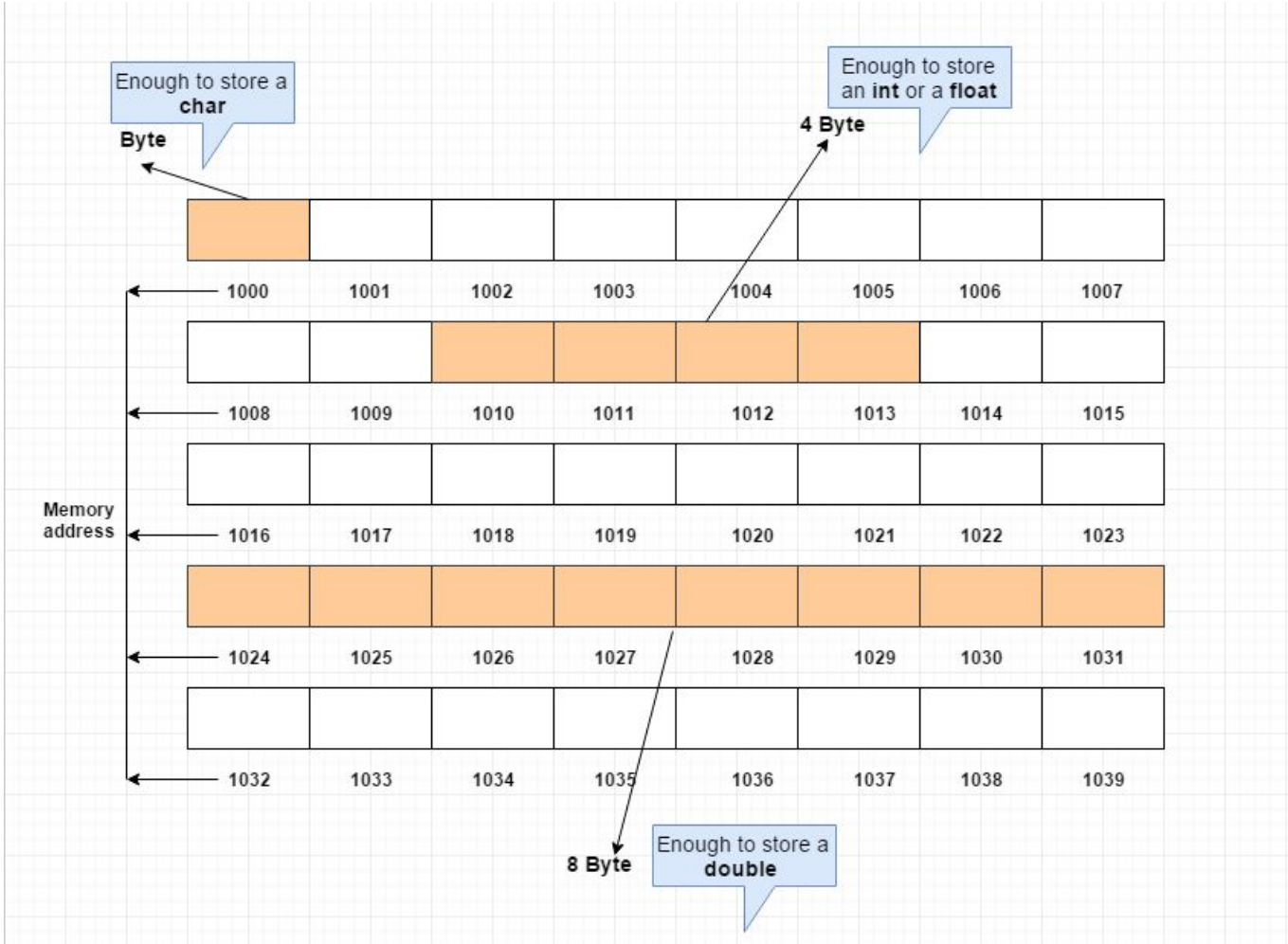


Desventajas

- Acceso a estructuras de datos fuera de límites.
- Referencia a variables automáticas después de su desaparición.
- Referencia a memoria asignada en el *heap* después de ser liberada.
- Desreferenciar un puntero antes de asignar memoria.

El uso incorrecto de punteros puede conducir a una serie de problemas potenciales, incluyendo errores de segmentación, corrupción de memoria y comportamiento impredecible del programa.

Memoria


Enough to store a char
Byte
Enough to store an int or a float
4 Byte
1000 1001 1002 1003 1004 1005 1006 1007
1008 1009 1010 1011 1012 1013 1014 1015
Memory address
1016 1017 1018 1019 1020 1021 1022 1023
1024 1025 1026 1027 1028 1029 1030 1031
1032 1033 1034 1035 1036 1037 1038 1039
8 Byte
Enough to store a double

Tamaño y Tipos de Punteros

- El tamaño de los punteros es un problema cuando nos preocupamos por la compatibilidad y portabilidad de la aplicación.
- En la mayoría de las plataformas modernas, el tamaño de un puntero a datos es normalmente el mismo, independientemente del tipo de puntero.
- Aunque el estándar de C no dicta que el tamaño sea el mismo para todos los tipos de datos, esto suele ser el caso. Sin embargo, el tamaño de un puntero a una función puede ser diferente al tamaño de un puntero a datos.

Tamaño y Tipos de Punteros

- Siempre utiliza el operador sizeof cuando se necesita conocer el tamaño de un puntero.

```
printf("Size of *char: %d\n",sizeof(char*));
```

64-bit data models

Data model	short int	int	long int	long long	Pointer, size_t	Sample operating systems
ILP32	16	**32**	**32**	64	**32**	x32 and arm64ilp32 ABIs on Linux systems; MIPS N32 ABI.
LLP64	16	32	32	**64**	**64**	Microsoft Windows (x86-64, IA-64, and ARM64) using Visual C++; and MinGW
LP64	16	32	**64**	**64**	**64**	Most Unix and Unix-like systems, e.g., Solaris, Linux, BSD, macOS. Windows when Cygwin; z/OS
ILP64	16	**64**	**64**	**64**	**64**	HAL Computer Systems port of Solaris to the SPARC64
SILP64	**64**	**64**	**64**	**64**	**64**	*Classic* UNICOS[45][46] (versus UNICOS/mp, etc.)

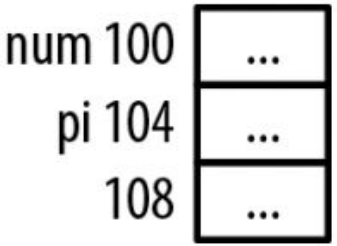
Data Type	Size in Bytes
byte	1
char	1
short	2
int	4
long	8
float	4
double	8

Declaración de Punteros

Las variables puntero se declaran utilizando un tipo de datos seguido de un asterisco y luego del nombre de la variable puntero.

```
int num; // declaración de entero

int *pi; // declaración de un puntero a entero
```



Los punteros a memoria no inicializada pueden ser un problema. Si se desreferencia un puntero de este tipo, es probable que el contenido del puntero no represente una dirección válida, y si lo hace, es posible que no contenga datos válidos. Una dirección inválida es aquella a la que el programa no está autorizado a acceder.

Cómo leer una declaración

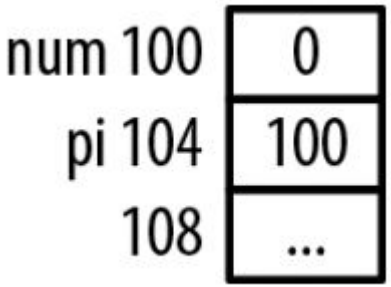
1. `pci` is a variable	`const int *pci;`
2. `pci` is a pointer variable	`const int *pci;`
3. `pci` is a pointer variable to an integer	`const int *pci;`
4. `pci` is a pointer variable to a constant integer	`const int *pci;`

& Operador de referencia (dirección)

El operador de referencia, **&**, devolverá la dirección de su operando.

```
num = 0;

int *pi = &num;
```



Es una buena práctica inicializar un puntero tan pronto como sea posible.

*, Operador de desreferencia(indirección)

El operador de indirección, *, devuelve el valor al que apunta una variable puntero. Esto se refiere comúnmente como **desreferenciar un puntero.**

```
int num = 5;

int *pi = &num;

printf("%p\n", *pi); // Muestra 5

*pi = 200;

printf("%d\n", num); // Muestra 200
```

Punteros a Funciones

Se puede declarar un puntero para que apunte a una función.

La notación de declaración es un poco críptica.

La función recibe void y devuelve void. El nombre del puntero es **foo:**

```
void (*foo)()
```

pi = NULL

Un puntero nulo y un puntero no inicializado son diferentes. Un puntero no inicializado puede contener cualquier valor, mientras que un puntero que contiene NULL no hace referencia a ninguna ubicación en la memoria.

```
pi = 0;
pi = NULL;
pi = 100;    // Syntax error
pi = num;    // Syntax error
```

```
if(pi) {
    // Not NULL
} else {
    // Is NULL
}
```

Un puntero nulo nunca debería ser desreferenciado porque no contiene una dirección válida. Al ejecutarse, provocará la terminación del programa.

Ejemplo:

```
int num;
int *pi = 0;
pi = &num;
*pi = 0;
```

Punteros a void

Un puntero a void es un puntero de **propósito general** utilizado para almacenar referencias a **cualquier tipo de datos**.

```
void *pv;
```

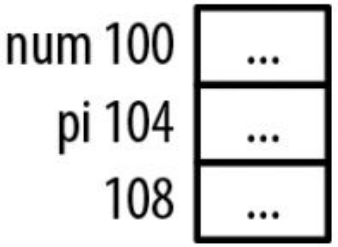
Tiene dos propiedades interesantes:

- Un puntero a void tendrá la misma representación y alineación de memoria que un puntero a char.
- Un puntero a void nunca será igual a otro puntero. Sin embargo, dos punteros a void asignados con el valor NULL serán iguales.

Punteros a void

Cualquier puntero puede asignarse a un puntero a void. Luego, puede ser convertido de vuelta a su tipo de puntero original. Cuando esto sucede, el valor será igual al valor original del puntero.

```c
int num;
int *pi = &num;
printf("Value of pi: %p\n", pi);
void* pv = pi;
pi = (int*) pv;
printf("Value of pi: %p\n", pi);
```



Punteros a void

```
size_t size = sizeof(void*);    // Legal
size_t size = sizeof(void);     // Illegal
```

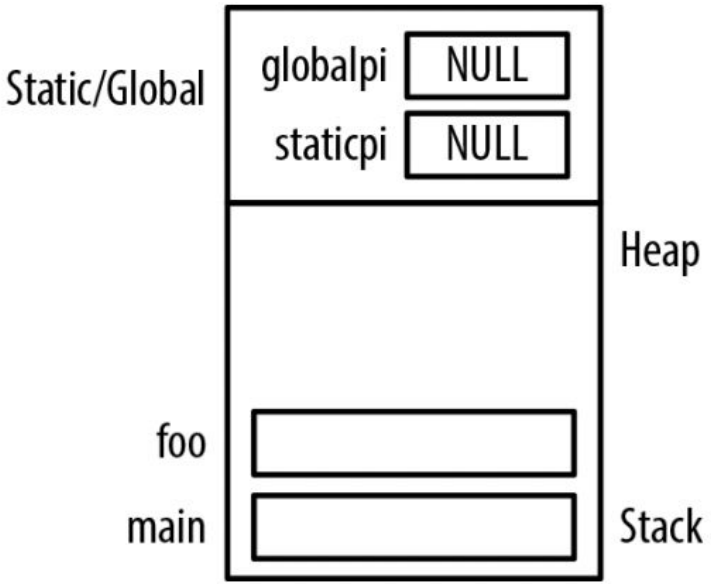
Punteros globales y estáticos

Si un puntero se declara como global o estático, se inicializa a NULL cuando el programa comienza.

```
int *globalpi;

void foo() {
   static int *staticpi;
   ...
}

int main() {
   ...
}
```



Operadores de Punteros

Operator	Name	Meaning
*		Used to declare a pointer
*	Dereference	Used to dereference a pointer
->	Point-to	Used to access fields of a structure referenced by a pointer
+	Addition	Used to increment a pointer
-	Subtraction	Used to decrement a pointer
== !=	Equality, inequality	Compares two pointers
> >= < <=	Greater than, greater than or equal, less than, less than or equal	Compares two pointers
(data type)	Cast	To change the type of pointer

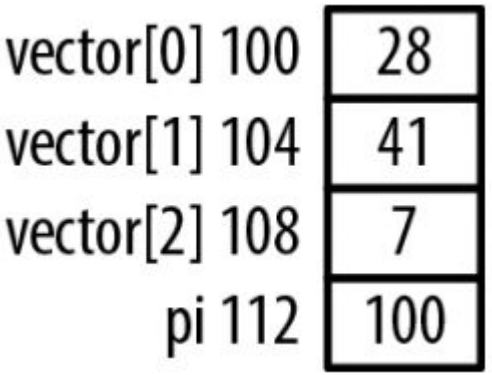
Aritmética de Punteros

Sumar un entero a un puntero

Esta operación es muy común y útil. Cuando sumamos un entero a un puntero, la cantidad agregada es el producto del entero por el número de bytes del tipo de datos subyacente.

```
int vector[] = {28, 41, 7};
int *pi = vector;       // pi: 100

printf("%d\n",*pi);     // Displays 28
pi += 1;                // pi: 104
printf("%d\n",*pi);     // Displays 41
pi += 1;                // pi: 108
printf("%d\n",*pi);     // Displays 7
```



Aritmética de Punteros

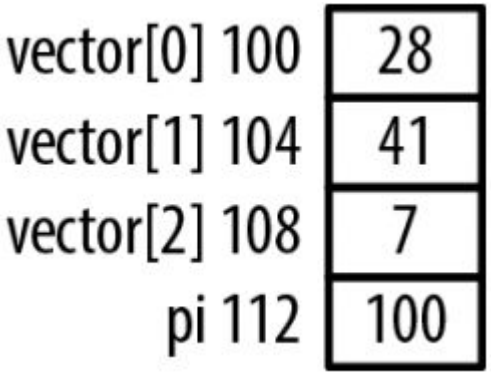
```
int num = 5;
void *pv = &num;
printf("%p\n",pv);
pv = pv+1;          //Syntax warning
```

Aritmética de Punteros

Restar un entero a un puntero

```
int vector[] = {28, 41, 7};
int *pi = vector + 2;  // pi: 108

printf("%d\n",*pi);    // Displays 7
pi--;                  // pi: 104
printf("%d\n",*pi);    // Displays 41
pi--;                  // pi: 100
printf("%d\n",*pi);    // Displays 28
```



Aritmética de Punteros

Restar dos punteros

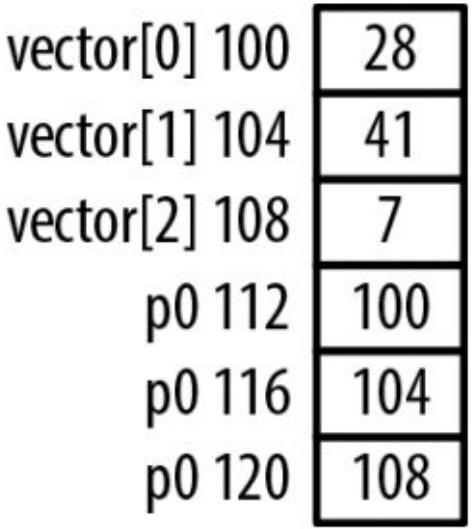
Cuando se resta un puntero de otro, obtenemos la diferencia entre sus direcciones. Esta diferencia normalmente no es muy útil, excepto para determinar el orden de los elementos en un arreglo.

La diferencia entre los punteros es el número de "unidades" por las cuales difieren. El signo de la diferencia depende del orden de los operandos

Aritmética de Punteros

```c
int vector[] = {28, 41, 7};
int *p0 = vector;
int *p1 = vector+1;
int *p2 = vector+2;

printf("p2-p0:  %d\n",p2-p0);     // p2-p0:  2
printf("p2-p1:  %d\n",p2-p1);     // p2-p1:  1
printf("p0-p1:  %d\n",p0-p1);     // p0-p1:  -1
```

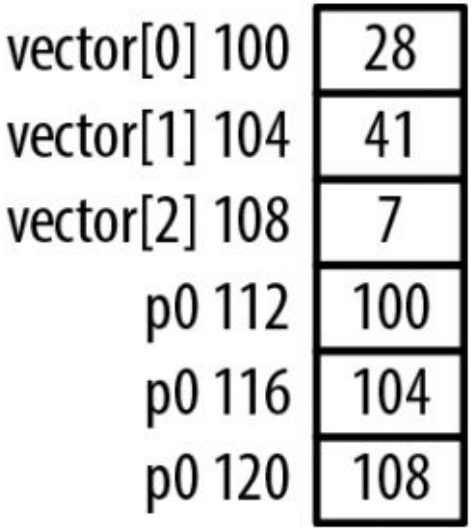


Aritmética de Punteros

Comparar punteros

```
int vector[] = {28, 41, 7};
int *p0 = vector;
int *p1 = vector+1;
int *p2 = vector+2;

printf("p2>p0:  %d\n",p2>p0);     // p2>p0:  1
printf("p2<p0:  %d\n",p2<p0);     // p2<p0:  0
printf("p0>p1:  %d\n",p0>p1);     // p0>p1:  0
```



Puntero a puntero

Los punteros pueden utilizar diferentes niveles de indirección. No es raro ver una variable declarada como **un puntero a un puntero**, a veces llamado un **puntero doble.**

```
char *titles[] = {"A Tale of Two Cities",
        "Wuthering Heights","Don Quixote",
        "Odyssey","Moby-Dick","Hamlet",
        "Gulliver's Travels"};
```

Puntero a puntero

```
char *titles[] = {"A Tale of Two Cities",
       "Wuthering Heights","Don Quixote",
       "Odyssey","Moby-Dick","Hamlet",
       "Gulliver's Travels"};
```

```
char **bestBooks[3];
char **englishBooks[4];
```

```
bestBooks[0] = &titles[0];
bestBooks[1] = &titles[3];
bestBooks[2] = &titles[5];

englishBooks[0] = &titles[0];
englishBooks[1] = &titles[1];
englishBooks[2] = &titles[5];
englishBooks[3] = &titles[6];

printf("%s\n",*englishBooks[1]);
```

Puntero a puntero

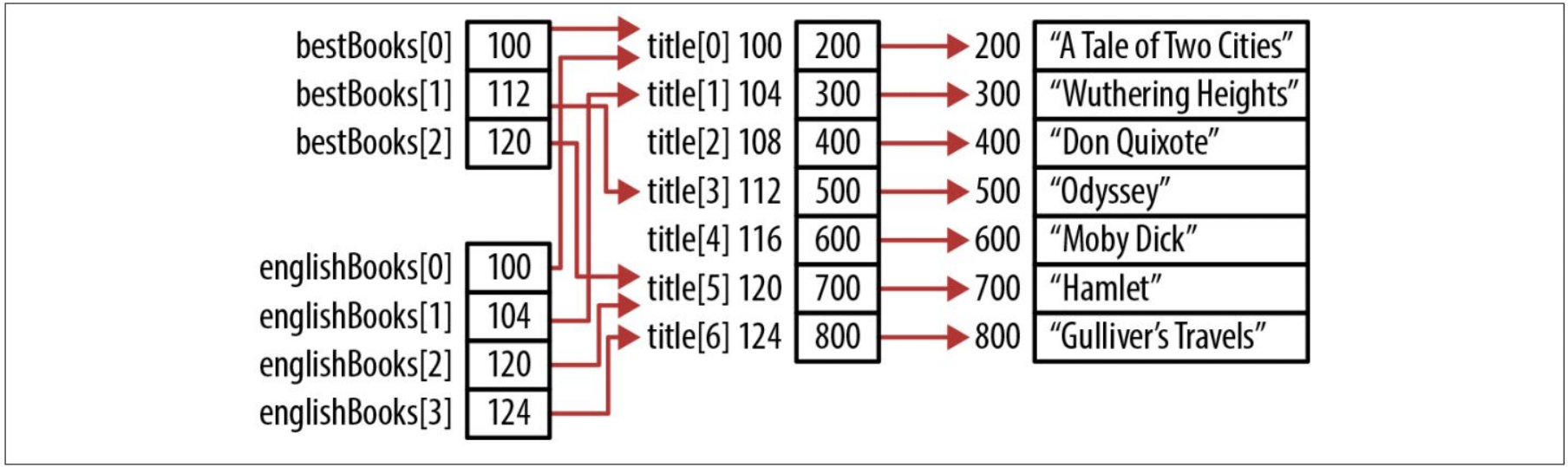


Figure 1-10. Pointers to pointers

Constantes y Punteros

Punteros a una constante

Se puede definir un puntero para que apunte a una constante. Esto significa que el puntero no puede ser utilizado para modificar el valor al que hace referencia

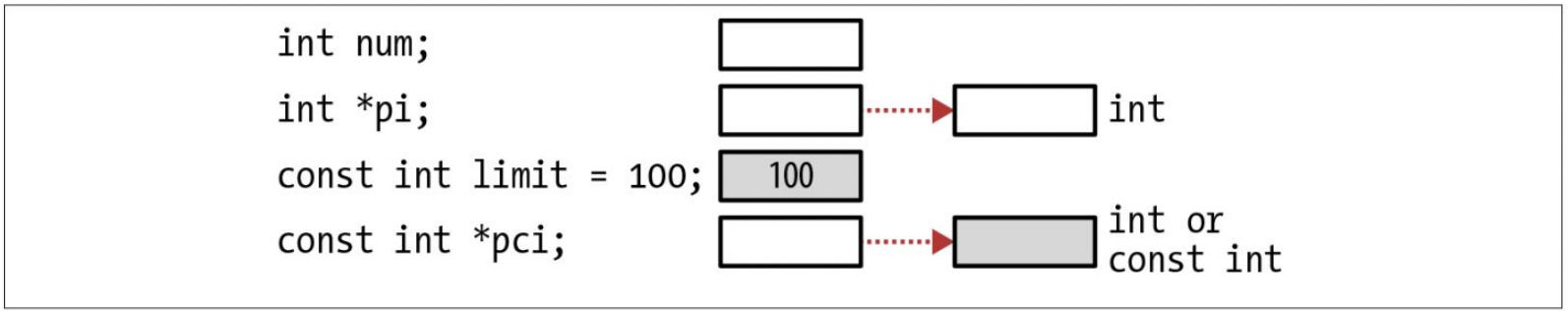


Figure 1-12. Pointer to a constant

Constantes y Punteros

Punteros a una constante

Se puede definir un puntero para que apunte a una constante. Esto significa que el puntero no puede ser utilizado para modificar el valor al que hace referencia

```
int num = 5;
const int limit = 500;
int *pi;                    // Pointer to an integer
const int *pci;             // Pointer to a constant integer

pi = &num;
pci = &limit;
```

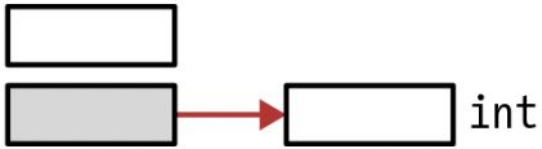
```
*pci = 200;
```

Constantes y Punteros

Punteros constantes a no constantes

Aunque el puntero no puede ser cambiado, los datos a los que apunta pueden ser modificados

```
int num;
int * const cpi = &num;
```

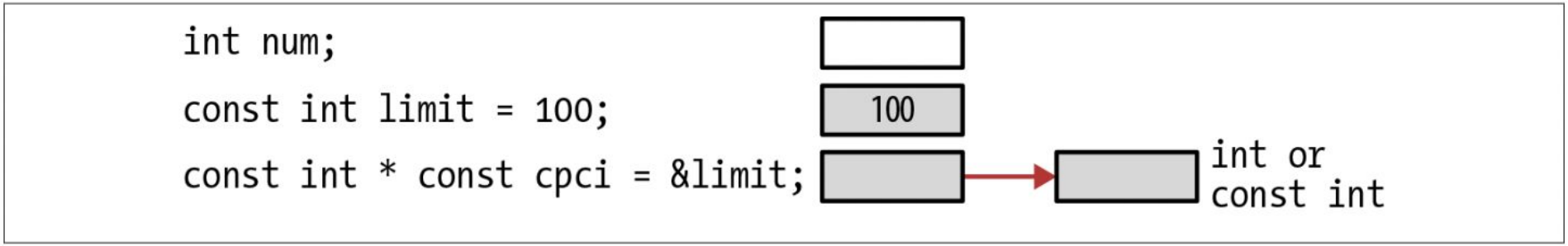


Constantes y Punteros

Punteros constantes a constantes

Un puntero constante a una constante es un tipo de puntero poco utilizado. El puntero no puede ser cambiado y los datos a los que apunta no pueden ser modificados a través del puntero.

```
const int * const cpci = &limit;
```



Constantes y Punteros

Los punteros a constantes también pueden tener varios niveles de indirección.

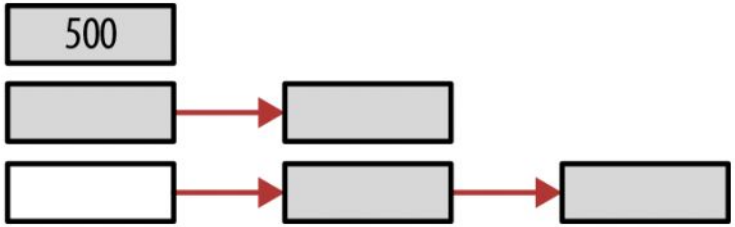
```
const int * const cpci = &limit;
const int * const * pcpci;
```

Constantes y Punteros

Los punteros a constantes también pueden tener varios niveles de indirección.

```
const int * const cpci = &limit;
const int * const * pcpci;
```

```
const int limit = 500;
const int * const cpci = &limit;
const int * const * pcpci = &cpci;
```



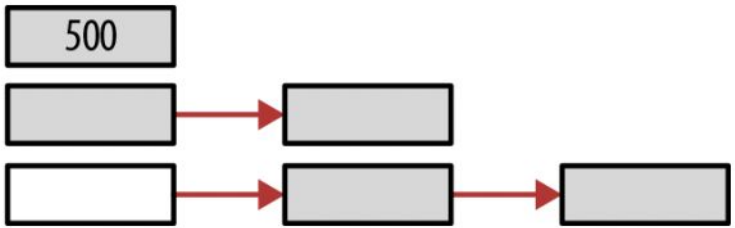
Constantes y Punteros

Los punteros a constantes también pueden tener varios niveles de indirección.

```
const int * const cpci = &limit;
const int * const * pcpci;

printf("%d\n",*cpci);
pcpci = &cpci;
printf("%d\n",**pcpci);
```

```
const int limit = 500;
const int * const cpci = &limit;
const int * const * pcpci = &cpci;
```



Constantes y Punteros

Pointer Type	Pointer Modifiable	Data Pointed to Modifiable
Pointer to a nonconstant	✓	✓
Pointer to a constant	✓	X
Constant pointer to a nonconstant	X	✓
Constant pointer to a constant	X	X

Introducción a punteros

Ciencia de la Computación II

MSc. Gina Muñoz Salas