

Algoritmos y

Estructuras

De

Datos

Un tipo de datos especial

TIPO PUNTERO

TIPOS DE DATOS



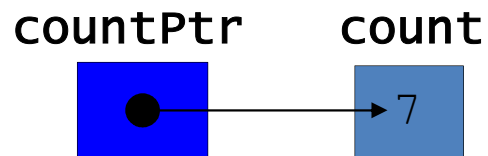
PUNTEROS

- Potentes, pero hay que dominar
- Simulan llamadas por referencia
- Estrecha relación con los arreglos y strings
- Para qué sirven?
 - Permiten el manejo dinámico de memoria
 - Permiten la implementación de estructuras de datos dinámicas
 - Permiten implementar arrays

Variables tipo Puntero

Contienen direcciones de memoria como sus valores.

- Las variables normales contienen un valor específico (**referencia directa**).
- Un **puntero** contiene la *dirección* de una variable que tiene un valor específico (**referencia indirecta**).



Definición e Inicialización de variables Puntero

▪ Definición de Punteros:

- Se usa * con variables puntero: **int *Punt1;**
 - Define un puntero a un **int** (puntero de tipo int *)
- Varios punteros requieren el uso de un * antes de cada definición de variable:
int *Punt1, *Punt2;
- Se pueden definir punteros a **cualquier tipo de datos.**

▪ Inicialización de Punteros:

- Los punteros se inicializan a 0, NULL, o a una dirección de memoria concreta.
 - 0 ó NULL indican que apunta a nada (mejor NULL).
- int *Punt1 = NULL;**
int *Punt1 = 0;

Operadores de Punteros

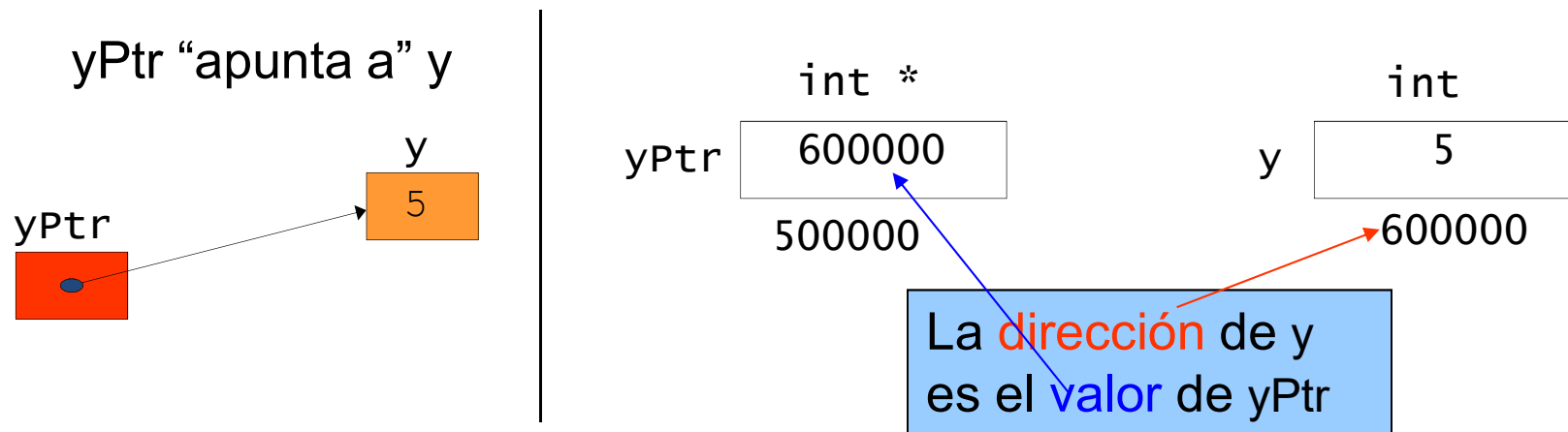
& (operador de dirección)

- Retorna la dirección del operando

```
int y = 5;
```

```
int *yPtr;
```

```
yPtr = &y; /* yPtr almacena la dirección de y */
```



Operadores de Puntero

* (operador de indirección/desreferencia)

- Retorna el contenido de *la variable cuya dirección está almacenada en* el puntero (operando de *), o

- ... *La variable a la que apunta* el puntero.

- *yPtr retorna 5 (porque yPtr apunta a y)

- * puede ser usado para asignaciones:

yPtr = 7; / cambia y al valor 7 */

- El puntero desreferenciado (operando de *) debe ser una variable a la izquierda del signo igual (no constante)

Ejemplo 1

```
#include <iostream>
using namespace std;

int main() {
    int a;           /* a es un entero */
    int* Ptr;        /* Ptr es un puntero a entero */
    a = 12;
    Ptr = &a;        /* Ptr inicializado con la direccion de a */

    cout << "La direcccion de a es : " << &a << endl
         << "El valor de Ptr es: " << Ptr;

    cout << endl << endl << endl;
    cout << "El valor de a es : " << a << endl
         << "El valor de la direccion que apunta Ptr es: " << *Ptr;

    cout << endl << endl << endl;
    cout << " * y & son complementarios: " << endl << endl;
    cout << " &*Ptr = " << &*Ptr;
    cout << endl;
    cout << " *&Ptr = " << *&Ptr;

    return 0;
}
```

La direcccion de a es : 0x71fefc

El valor de Ptr es: 0x71fefc

El valor de a es : 12

El valor de la direccion a la que apunta Ptr es: 12

* y & son complementarios:

&*Ptr = 0x71fefc

*&Ptr = 0x71fefc

Expresiones con punteros y Aritmética de punteros

- Sobre punteros se pueden realizar operaciones aritméticas:
 - Incrementar/decrementar un puntero (++ , --)
 - Sumar/restar un entero a un puntero (+ , += , - , -=)

Esto implica que la dirección almacenada en la variable tipo puntero se incrementa (o decrementa) en tantos bytes como corresponde al tipo de datos apuntado.

- Los punteros se pueden restar entre ellos

Estas operaciones tienen significado cuando se apunta a un arreglo.

Ejemplo 2

```
#include <iostream>
using namespace std;

int main() {
    int a, *p;
    double b, *q;

    a = 5;
    p = &a;
    b = 35.89;
    q = &b;

    cout << "p: " << p << endl;
    cout << "q: " << q << endl;
    cout << endl;
    cout << "*p: " << *p << endl;
    cout << "*q: " << *q << endl;
    cout << endl;
    p++; q++;
    cout << "p: " << p << endl;
    cout << "q: " << q << endl;
    cout << endl;
    cout << "*p: " << *p << endl;
    cout << "*q: " << *q << endl;
    return 0;
}
```

```
p: 0x28ff14
q: 0x28ff08

*p: 5
*q: 35.89

p: 0x28ff18
q: 0x28ff10

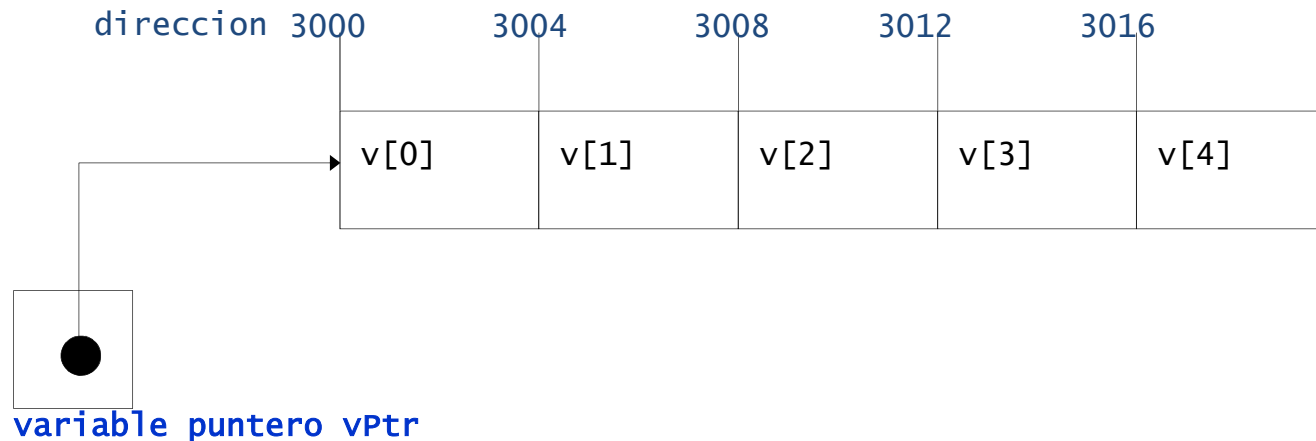
*p: 2686736
*q: 1.15794e-313
```

Enteros: 4 bytes

Doubles: 8 bytes

Aritmética de punteros en arreglos

- **Arreglo v** de 5 elementos **int** (int de 4 bytes)
 - **vPtr** apunta al primer elemento **v[0]**
 - En la posición 3000, **vPtr = 3000**
 - **vPtr += 2;** apunta vPtr a la posición 3008
 - **vPtr** apunta **a v[2]** (incrementado en 2), como la máquina tiene enteros de 4 bytes, **vPtr** apunta a la dirección **3008**.



Ejemplo 3: Suma de entero a puntero

```
#include <iostream>
using namespace std;

int main(int argc, char *argv[]) {
    int v[5]={31,28,17,4,9};

    int *vPtr;
    vPtr = v;
    cout << *vPtr << endl;

    vPtr += 2;
    cout << *vPtr << endl;

    return 0;
}
```

```
31
17
```

Asignación de punteros

Expresiones con punteros y Aritmética de punteros

Resta de punteros

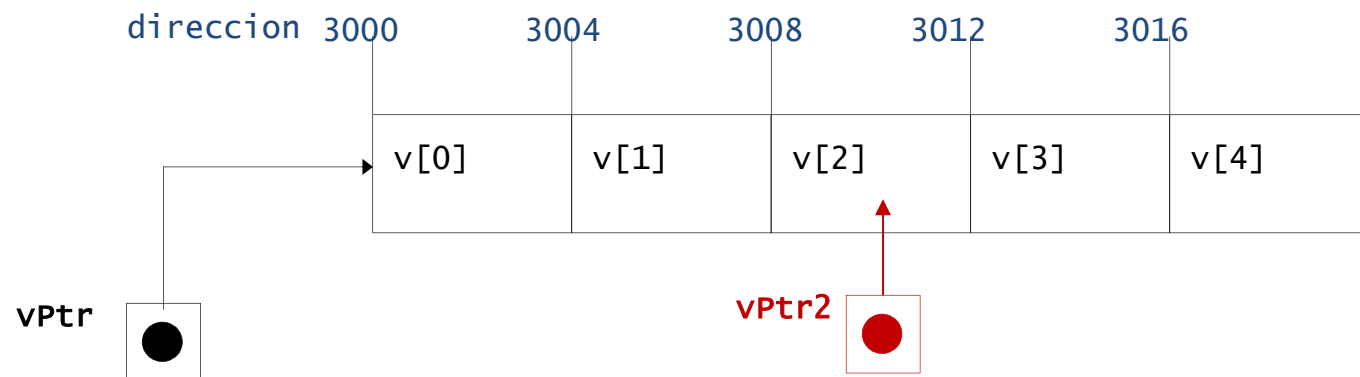
Retorna el número de elementos entre uno y otro.

– Si:

$\text{vPtr2} = \&\text{v}[2];$

$\text{vPtr} = \&\text{v}[0];$

– **$\text{vPtr2} - \text{vPtr}$** retorna **2**



Ejemplo 4: Resta de punteros

```
#include <iostream>
using namespace std;

int main(int argc, char *argv[]) {
    int v[5]={31,28,17,4,9};

    int *vPtr;
    vPtr = v;

    int *vPtr2 = &v[2];

    cout << vPtr2 - vPtr << endl;

    cout << endl << *vPtr ;
    cout << endl << *vPtr2 ;
    vPtr2--;
    cout << endl << *vPtr2 ;

    return 0;
}
```

```
2
31
17
28
```

Uso de notación de subíndices y punteros con arreglos

Ejemplo 5

```
#include <iostream>
using namespace std;
```

```
int main(){
    int b[5] = {1,3,5,7,9};
    int* p = b;
    int i;

    cout << "Salida con indices" << endl;
    for (i=0; i<5; i++)
        cout << b[i] << " ";

    cout << endl << endl;
    cout << "Salida con punteros" << endl;
    for (i=0; i<5; i++)
        cout << *(p+i) << " ";

    cout << endl << endl;
    cout << "Salida con punteros alternativa" << endl;
    for (i=0; i<5; i++)
        cout << p[i] << " ";

    cout << endl << endl;
    cout << "Salida con vector como puntero" << endl;
    for (i=0; i<5; i++)
        cout << *(b+i) << " ";

    return 0;
}
```

Salida con indices

1 3 5 7 9

Salida con punteros

1 3 5 7 9

Salida con punteros alternativa

1 3 5 7 9

Salida con vector como puntero

1 3 5 7 9

Ejemplo 6

```
#include <iostream>
using namespace std;

int main() {
    int t [2][3] = {{1,2,3},{4,5,6}};
    int* z = t[0];

    cout << z << endl;
    cout << &t[0][0] << endl;
    cout << "-----" << endl;

    cout << t[1][0] << endl;
    cout << *(z+3) << endl;
    cout << "-----" << endl;
    cout << &t[1][0] << endl;
    cout << z+3;

    return 0;
}
```

0x28ff04
0x28ff04

4
4

0x28ff10
0x28ff10

t[0] indica la dirección de
memoria de la 1era fila

LEER

Capítulo 12

“Apuntadores”

**Libro: “C++ para ingeniería y ciencias” 2da edición -
Gary J. Bronson, pág. 667**