Algoritmos y

Estructuras

De

Datos

Un tipo de datos especial…

TIPO PUNTERO

TIPOS DE DATOS

Tipos de Datos

- **Simples o básicos** — Valor único (indivisible)
  - Caracteres
  - Enteros
  - Reales
  - Booleanos
- **Punteros**
- **Estructurados** — Colección de valores relacionados.

  Se caracterizan por:
  - *el tipo de dato de sus elementos
  - * la forma de almacenamiento
  - * la forma de acceso

  - **Estructuras estáticas**: ARREGLOS (tablas), CADENAS, TUPLAS
  - **Estructuras dinámicas**: LISTAS, PILAS, COLAS, ÁRBOLES, GRAFOS

PUNTEROS

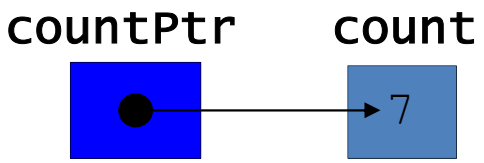
- Potentes, pero hay que dominar
- Simulan llamadas por referencia
- Estrecha relación con los arreglos y strings
- Para qué sirven?
  - Permiten el manejo dinámico de memoria
  - Permiten la implementación de estructuras de datos dinámicas
  - Permiten implementar arrays

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;

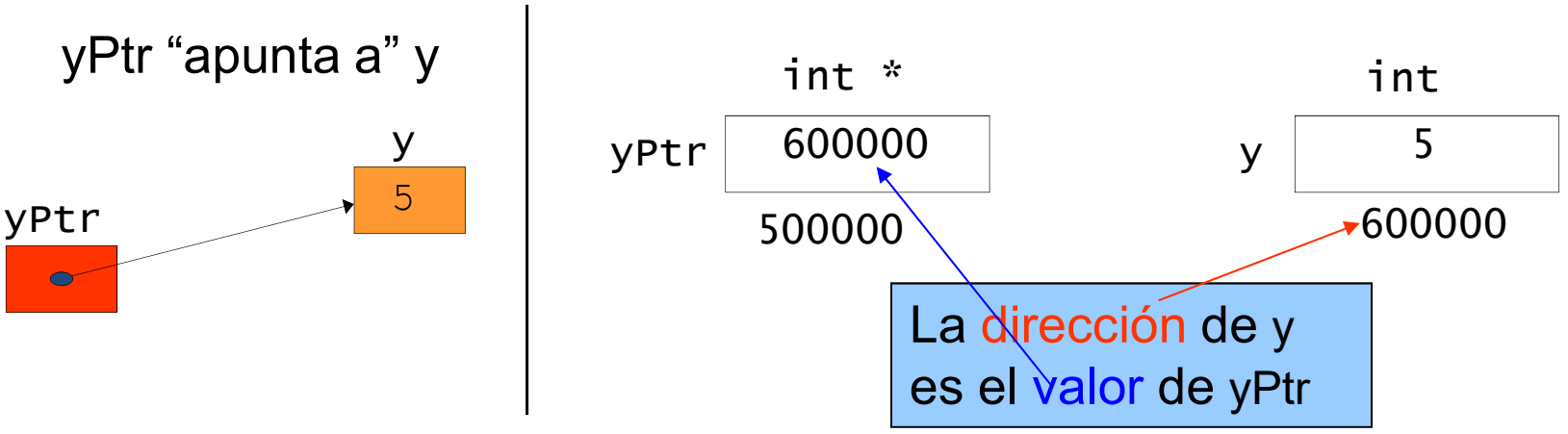
      int *Punt1 = 0;

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 (no constante) a la izquierda del signo igual

Ejemplo 1

```
La direccion 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
```

```
#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 direccion 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;
}
```

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 sentido cuando se apunta a un arreglo.

Ejemplo 2

```cpp
#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;
}
```

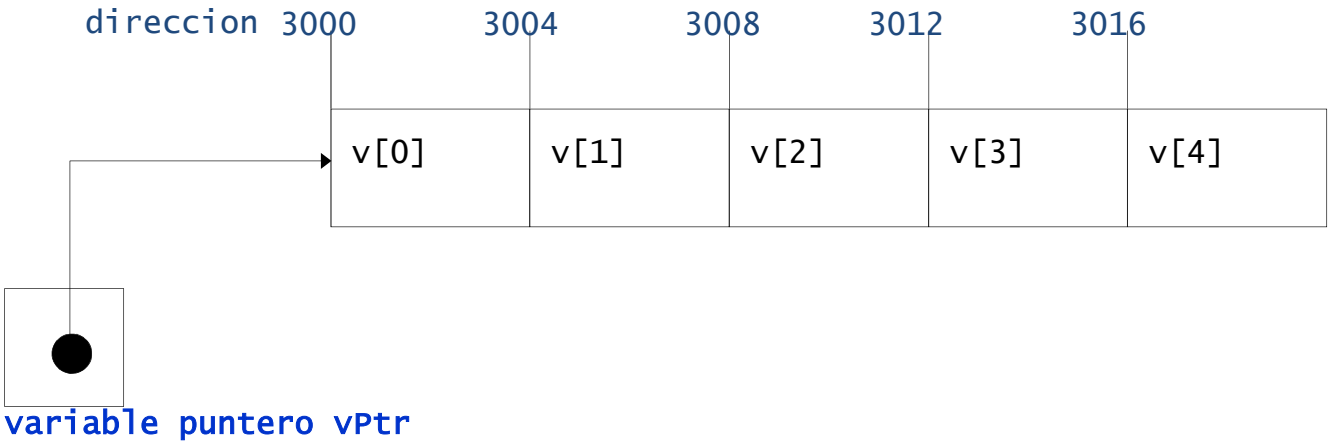
Enteros: 4 bytes

```
p: 0x71ff04
q: 0x71fef8

*p: 5
*q: 35.89

p: 0x71ff08
q: 0x71ff00

*p: 7470848
*q: 1.16009e-313
```

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;
}
```

Asignación de punteros

```
31
17
```

Expresiones con punteros y Aritmética de punteros

Resta de punteros

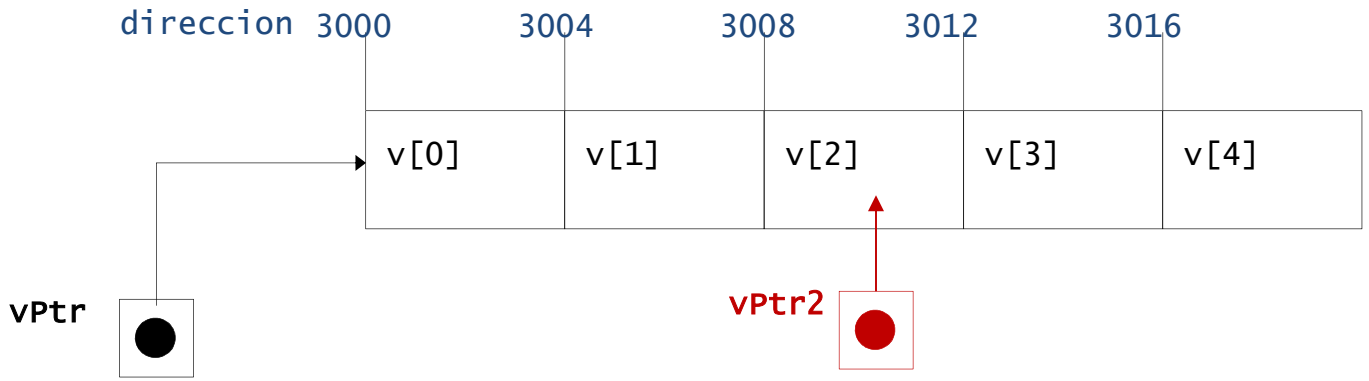
Retorna el número de elementos entre uno y otro.

- Si:

  vPtr2 = &v[2];

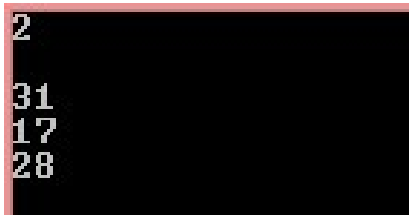
  vPtr = &v[0];

- **vPtr2 - 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;
}
```



Expresiones con punteros y Aritmética de punteros

- Los punteros del mismo tipo se pueden asignar entre ellos. Ejemplo:

  ptr = mi_arreglo;

  La direccion del 1er elemento de *mi_arreglo* se asigna al puntero *ptr*, y de esta manera ambos son equivalentes.

  – Si no son del mismo tipo, se debe usar un operador **cast**.

  – Excepción: **punteros sin tipo** (genéricos)

  void * Ptr; /* Declara un puntero genérico */

  - **Puntero genérico:** no se asigna a ningún tipo de dato específico, sino que apunta a cualquier tipo de dato.
  - No se necesita casting para convertir cualquier puntero a un puntero void.
  - Los punteros void no pueden ser desreferenciados.

La relación entre punteros y arreglos

- Los arreglos y punteros se relacionan estrechamente
  - **Un nombre de arreglo es un puntero:** contiene la dirección del primer elemento del arreglo.
  - **Un nombre de arreglo es un puntero constante:** no se puede modificar, sólo se puede utilizar para acceder a los elementos del arreglo.
  - Los punteros se pueden utilizar para acceder a un arreglo mediante subíndices.
- Dado un arreglo **b[5]** y un puntero **bPtr**
  - Se pueden hacer iguales mediante:

    bPtr = b;

    - El nombre del arreglo (b) es realmente la dirección del primer elemento del arreglo b[5].
    - bPtr = &b[0]; explícitamente asigna a bPtr la dirección del primer elemento de b.

La relación entre punteros y arreglos

- **Elemento b[3]**
  - Puede ser accedido mediante *(bPtr + 3)
    - Donde 3 es el desplazamiento.
    - Notación puntero/desplazamiento
  - También puede ser accedido mediante bptr[3]
    - Notación puntero/subíndice
    - bPtr[3] es lo mismo que b[3]
  - O, puede ser accedido mediante aritmética de punteros sobre el arreglo:

    *(b + 3)

Uso de notación de subíndices y punteros con arreglos

Ejemplo 5

```cpp
#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