Matrices y Punteros

Antonio Espín Herranz

Matrices y punteros

- Punteros: Conceptos, uso, aritmética de punteros en C.
- Matrices: Conceptos y manipulación.
- Matrices de varias dimensiones.
- Matrices como punteros y punteros a matrices.
- Punteros constantes y punteros a constantes.
- Paso de matrices como parámetros.
- Los punteros como parámetros.
- Punteros a funciones.
- Punteros a punteros.

Punteros, conceptos, uso y aritmética

- Un puntero es una variable cuyo contenido es la dirección de una variable de cualquier tipo.
- Declaración:
  - tipo *nombre_puntero;
  - int *p; // Apunta a variables de tipo entero.
  - Un puntero solo puede apuntar a variables que sean de su propio tipo.
- Operadores:
  - * Devuelve o accede al contenido del puntero.
  - & Devuelve la dirección de memoria de una variable.

Punteros, conceptos, uso y aritmética

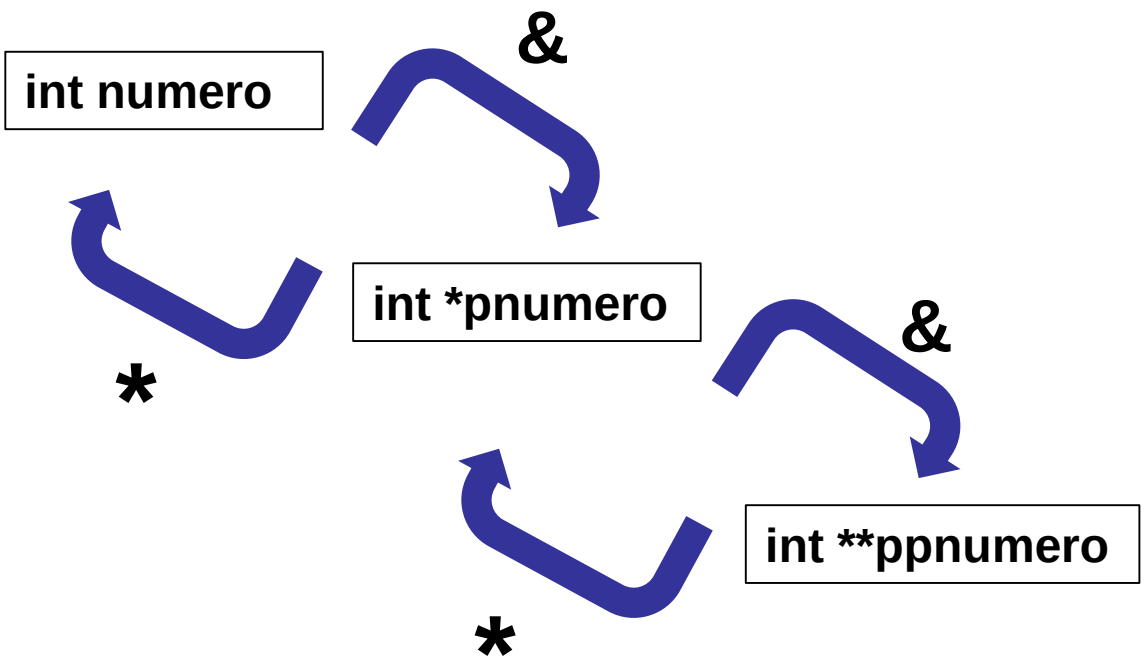
- Ejemplo:

```
int numero = 6;
int *p;
p = &numero;
*p = 10; // Se modifica el contenido de numero
     a partir de su dirección.
```

  - Los punteros los podemos inicializar a NULL. Ejemplo p = NULL;

  - En C, las variables puntero tienen que apuntar realmente a variables que sean del mismo tipo que el puntero.

Operadores & *



Mas operadores

- Relacionales: ==, >, >=, <, <=, !=
  - Podemos comparar las direcciones de memoria o el contenido de las direcciones.
  - int x = 3, y = 5, *p, *q; p = &x; q = &y;
  - if (p == q) // Comparando las direcciones.
  - if (p > q) // Las direcciones de memoria.
  - if (*p != *q) // El contenido, 3 != 5.

Operadores e impresión de punteros

- De incremento: ++ --
  Podemos modificar el contenido de la variable:
  (*p)++; // Forzamos a acceder al contenido.
  *(p++); // Apuntará a la siguiente dirección de su tipo base.
  *(p + 1) // Idem del anterior pero sin modificar p.
  En caso de ser un entero se le sumara 2 bytes.

- Con los punteros NO podemos:
  - Sumar dos punteros, multiplicar o dividir.

- La diferencia de punteros SI se permite:
  - Al restar las dos direcciones (si están dentro de un array podemos obtener la posición, el índice).

- Impresión:
  int n = 0, *p;
  printf("n = %d, p = **%p**", n, **p**);

Arrays

- El nombre del array representa un puntero.

```
void imprimir(int *v, int numElems){
    // … Imprime el array.
    // OJO, aquí el numero de elementos NO se calcularía bien
    con sizeof, porque sizeof(v) daría lo que ocupa el puntero.
}

int main(){
    int array[] = {1,3,4,7,7,8,9};
    int tam = sizeof(array)/sizeof(int);
    imprimir(array, tam);
}
```

Matrices: conceptos y manipulación

- El nombre de un array representa un puntero.
- En el caso de los punteros se pueden modificar, en el caso de los array no se pueden modificar.
- Manipulación:

```
char str[100], *cad = NULL;
cad = str;   // También: cad = &str[0];
```

Matrices: conceptos y manipulación

```
// Impresión del contenido:
printf("%s %s", str, cad);
// Impresión de las direcciones:
printf("%p %p", str, cad);

// Distintas formas de almacenar una 'a' en el 3º carácter:
a)  str[2] = 'a';
b)  cad += 2;  *cad = 'a'; // No es muy usado, modifica el
    valor de cad.
c)  *(cad+2)='a';   // También vale: *(str+2)='a';
d)  cad[2] = 'a';
```

Matrices de varias dimensiones

- Podemos tener arrays de varias dimensiones.
- Definición: <tipo> nombre [nFilas][nCols];
- Definición con inicialización:
  - int tabla[2][3] = {51, 52, 53, 54, 55, 56};
  - int tabla[2][3] = { {51, 52, 53}, {54, 55, 56} };
  - Se puede dejar vacía la primera dimensión:
    - Int tabla[][3] = { {1,2,3}, {4,5,6}}; // El compilador cuenta las filas.
- Acceso a los elementos:
  - tabla[0][0] = 99; printf("%d", tabla[1][0]);
- Recorrer todos los elementos:
  - Utilizaremos un bucle **for** por cada dimensión. Máximo 256 dim

Matrices como punteros y punteros a matrices

- El nombre del array es un puntero.
- Arrays de punteros: int *p[10]; // Tenemos 10 punteros a enteros.
- int *ptr1[]; // Array de punteros a int.

- Punteros a arrays: tipo_elem_array (*puntero)[ ];
  - int (*ptr2)[]; // Puntero a un array de elementos int.
  - int *(*ptr3)[]; // Puntero a un array de punteros a int.

Punteros a array I

- Para pasar un matriz a una función, podemos hacer:

```
void imprimir(int n, int m, int tabla[n][m]){
    // Imprime la matriz …
}

void imprimir2(int n, int m, int (*tabla)[m]){
    // Imprime la matriz …recibe el puntero al array. m
    representa el número de columnas.
}
```

Punteros a array II

- En el main se declara la matriz, y para llamar a la funciones es suficiente con el nombre de la matriz.

```
int main(){
    int tabla[4][6];
    // … se carga de números  la tabla.
    imprimir(4,6,tabla);
    imprimir2(4,6,tabla)
}
```

Punteros constantes y punteros a constantes

- Punteros constantes:
  - <tipo de dato> *const <nombre puntero> = <dirección>
  - int x, int *const p1 = &x;
  - p1 es un puntero constante, pero *p1 es una variable, se puede cambiar el contenido de p1 pero no podemos alterar p1, es decir, el puntero.
  - char *const nombre = “luis”;
  - *nombre = ‘c’; // Esta modificando el 1º carácter.
  - nombre = &Otra_cadena; // ERROR.

Punteros constantes y punteros a constantes

- Punteros a constantes:
  - const <tipo de dato> *nombre_puntero = dirección de constante.
  - const int x = 25;
  - const int y = 50;
  - const int *p1 = &x;
  - p1 se define como un puntero a la constante x. los datos son constantes y no el puntero, p1 puede apuntar a otra constante.
  - p1 = &y;
  - Pero no podemos modificar el contenido de p1.
  - *p1 = 15; // ERROR.

Punteros constantes a constantes

- const <tipo de dato> *const <nombre_puntero> = constante;

  const int x = 25;

  const int *const p1 = &x;

- No podemos modificar ni p1 ni el contenido de p1.

  p1 = &otra_variable; // ERROR.

  *p1 = 56; // ERROR.

Paso de matrices como parámetros

- Cuando pasamos un array como parámetro a una función tenemos que tener en cuenta que la función receptora no conoce el número de elementos del array.

```
// Lo podemos recibir como un puntero, pero no sabemos la longitud.
int sumaDeEnteros(int *); // Prototipo.

// Se puede pasar como un array y además indicar el número de elementos en otro parámetro.
int sumaDeEnteros(int [], int);

// Llamada:
int numeros[5] = {2,3,4,5,6};
sumaDeEnteros(numeros);        // Se le pasa el nombre.
sumaDeEnteros(numeros, 5);     // Se le pasa el nombre y número de elementos.
```

Los punteros como parámetros

- Es una forma de pasar los parámetros por referencia y poder modificar el contenido de las variables.
- Si queremos modificar una variable en un procedimiento tendremos que mandar la dirección de dicha variable.
- Si queremos modificar la dirección de un puntero, tendremos que mandar ese puntero por referencia, es decir, recibir un puntero a puntero.

Los punteros como parámetros

Paso de variables por referencia:

```
void intercambiar(int *a, int *b){
    int aux;
    aux=*a; *a = *b; *b = aux;
}

void main(void){
    int x = 3, y = 5;
    intercambiar(&x, &y);
}
```

Paso de punteros por referencia:

```
void intercambiar(int **a, int **b){
    int *aux;
    aux=*a; *a = *b; *b = aux;
}

void main(void){
    int x = 3, y = 5, *px, *py;

    px = &x;  py = &y;
    intercambiar(&px, &py);
}
```

Punteros a funciones

- Podemos tener punteros que apunten a la dirección en memoria donde se encuentra una función.
- La dirección de la función viene indicada por el nombre de la misma.
- Tipo_de_retorno (*PunteroFuncion)(<lista parametros)
  - int f(int); // Define la función f.
  - int (*pf)(int) // Define puntero pf a función con argumento int.
  - pf = f; // Asigna la dirección de f a pf

Asignación de la función al puntero

- PunteroFuncion = unaFuncion;
- Ejemplo:

```
double calculo(int *v, unsigned n); // prototipo de la función.
double (*qf)(int *, unsigned); // Puntero a función.
int r[11] = { 3, 5, 6, 7, 1, 7, 3, 34, 5, 11, 44 };
double x;
qf =calculo;
x = qf(r, 11);  // Llamada a la función a partir del puntero.
```

Punteros a punteros

- Trabajaremos con un * mas.
- Definición: <tipo> **<nombre_puntero>;
- Ejemplos:
  - char **p = NULL; // Se trata de un puntero a punteros a carácter.
  - char *nombres[] = { “tomas”, “ana”};
  - p = nombres;
  - printf(“%s, %c”, *p, **p);
  - **Salida → tomas, t**

Ejemplos de prototipos

- int *fp(void); → Una función que devuelve un puntero a entero y no tiene parámetros.
- int (*fp)(void); → Puntero a una función que devuelve un entero y no tiene parámetros.
- int **fp(void); → Función que no tiene parámetros y que devuelve un puntero a puntero a entero.
- int (*fp(void))(int); → fp es una función que no tiene parámetros y que devuelve un puntero a una función que devuelve un entero y recibe un entero.
- int *(*fp)(void); → puntero a una función que devuelve un puntero a un entero y no tiene parámetros.
- int (*(*pf)(void))(int) → pf es un puntero a una función que no tiene parámetros y que devuelve un puntero a una función, que devuelve un entero y recibe un entero.