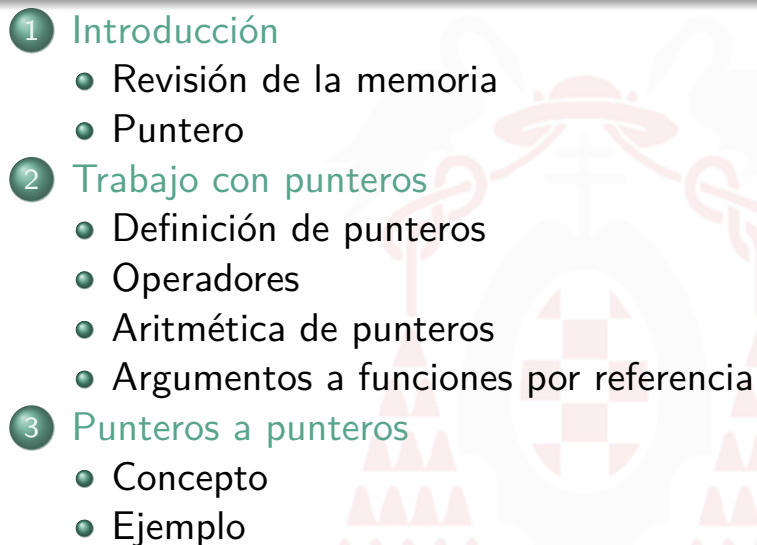


Tema 8: Punteros

Departamento de Automática
Universidad de Alcalá



Índice

- 
- 1 **Introducción**
 - Revisión de la memoria
 - Puntero
 - 2 **Trabajo con punteros**
 - Definición de punteros
 - Operadores
 - Aritmética de punteros
 - Argumentos a funciones por referencia
 - 3 **Punteros a punteros**
 - Concepto
 - Ejemplo

Introducción

Revisión de la memoria

- Von-Neumann define

- UAL
- Unidad de control
- Memoria

- La memoria contiene

- Código
- Datos

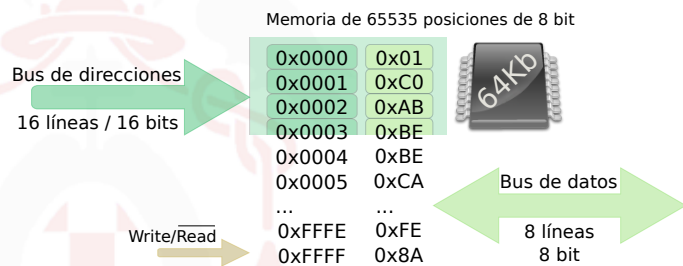
- Cada posición se identifica con una **dirección**

- Una dirección indica un byte (normalmente)

- Una dirección es un número

- Normalmente del tamaño de una palabra
- 8, 16, 32 ó 64 bits, según la arquitectura

- C permite trabajar con direcciones de memoria: *punteros*



Introducción

Puntero

- Una variable se guarda en una posición de memoria

- Permanece fija en esa posición
- La dirección se obtiene con el operador de indirección (&)

- Un **puntero** contiene una dirección de memoria

- Normalmente la dirección de una variable
- Es un tipo de variable
- Se como una variable ... con alguna diferencia

```
char variable=1;
char *puntero = &variable;
```

| | |
|---------|------|
| 0x0000 | 0x01 |
| 0x0001 | 0xFF |
| 0x0002 | 0xFE |
| 0x0003 | 0xBE |
| 0x0004 | 0xBE |
| 0x0005 | 0xCA |
| ... | ... |
| 0xFFFFE | 0x01 |
| 0xFFFF | 0x8A |

Trabajo con punteros

Definición de punteros

- Los punteros son variables
 - Se definen como variables
 - “*” indica que es puntero
 - Tipo de variable que apunta
- Un valor especial: NULL
 - Se lee puntero a null (o nulo)
 - Apunta a ningún sitio
- Punteros a void

Definición

```
tipo *nombre;
```

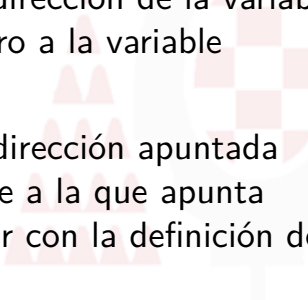
Ejemplo

```
int *puntero1;  
float *puntero2=NULL;  
char *puntero3;
```

```
puntero1 = NULL;
puntero2 = 0x01;
```

Trabajo con punteros

Operadores

- Dos operadores:
 - Indirección y dirección
- Indirección: "&"
 - Devuelve la dirección de la variable
 - Da un puntero a la variable
- Dirección: "*" 
 - Accede a la dirección apuntada
 - Da la variable a la que apunta
 - ¡No confundir con la definición de puntero!
- ¡Cuidado con los punteros!

Ejemplo

```
int *p;  
int a=0, b;  
  
p = &a;  
b = *p;  
  
printf(" %d", a);  
printf(" %d", b);  
printf(" %d", *p);
```

Trabajo con punteros

Aritmética de punteros

- Los operadores aritméticos pueden usarse con punteros
 - El significado cambia
- Si X apunta a una variable
 - $X+1$ apunta a la siguiente variable
 - El $+1$ suma el tamaño de la variable apuntada
- *Los vectores son punteros*
 - `int v[]` e `int *v` son equivalentes
 - `vector[3]` es igual a `*(vector+3)`

Ejemplo

```
int vector = {1,2,3};
int *p;

p = vector;
printf(" %d", *p);
printf(" %d", *(p+1));
printf(" %d", *(++p));
printf(" %d", p[1]);

p = &vector[0];
printf(" %d", p[1]);
```

Trabajo con punteros

Argumentos a funciones por referencia (I)

- Las variables en una función son locales
 - Incluso los argumentos
 - Los cambios no se traspasan
- Solución:
 - Pasar un puntero
 - Sigue siendo local
 - Cambia el objeto apuntado
- Los vectores son punteros
 - Siempre por referencia

Ejemplo incorrecto

```
void swap(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
}
```

Ejemplo correcto

```
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

Trabajo con punteros

Argumentos a funciones por referencia (II)

Ejemplo 1: Notación vectorial

```
void strcpy(char s[], char t[]) {  
    int i;  
    i = 0;  
    while ((s[i] = t[i]) != '\0')  
        i++;  
}
```

Ejemplo 2: Notación de punteros

```
void strcpy(char s*, char t*) {  
    while ((*s = *t) != '\0') {  
        s++;  
        t++;  
    }  
}
```



Trabajo con punteros

Argumentos a funciones por referencia (III)

Ejemplo2

```
int strlen(char *s) {  
    int n;  
  
    for (n=0; *s != '\0'; s++)  
        n++;  
  
    return n;  
}
```

Punteros a punteros

Concepto

- Los punteros son variables
 - Están en memoria
 - La dirección de un puntero es un puntero a un puntero
- Problemas que resuelven
 - Guardar varias cadenas
 - Modificar un puntero en una función
- *Las matrices son punteros a punteros*
 - `int m[] []` e `int **m` son equivalentes

Definición

```
int **m1;
int *m2[100];
int m3[][] =
    {{1, 2, 3},
     {2, 3, 4},
     {5, 6, 7}};
```

Punteros a punteros

Ejemplo

Ejemplo

```
int main(int argc, char *argv[]) {
    int i;
    for (i=0; i<argc; i++){
        printf("%s\n", argv[i]);
    }
}
```