El lenguaje C. Punteros y Arrays

Luis Llopis, 2024.

Dpto. Lenguajes y Ciencias de la Computación.

University of Málaga

UNIVERSIDAD
DE MÁLAGA | uma.es

El tipo Puntero. Por qué es importante

Poder y Flexibilidad

Código eficiente y flexible

El tipo Puntero. Por qué es importante

1. Acceso directo a la memoria
2. **Manipulación eficiente de arrays y cadenas de caracteres**
3. Paso por referencia
4. Creación de estructuras de datos complejas
5. Gestión dinámica de la memoria

Punteros y Arrays

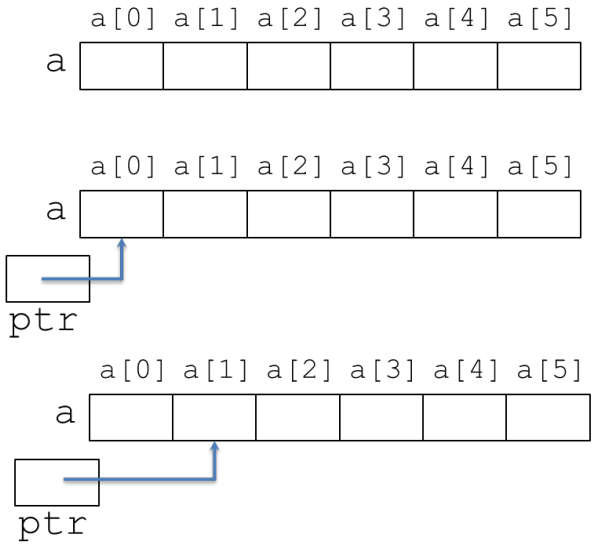
- En C, los punteros y los arrays están estrechamente relacionados
- Toda operación sobre un array se puede hacer con punteros
  - **Ventaja**: mayor control
  - **Inconveniente**: código más complejo para los que no dominan el lenguaje

Punteros y Arrays

```
int a[6] ;


int *ptr ;
ptr = &a[0] ;


ptr += 1 ;
```



Punteros y Arrays

```
a == &a[0]
```

```
*(ptr + 1) == a[1]
```

```
ptr++
```

Mueve el puntero a la siguiente posición del array

```
ptr += i
```

Mueve el puntero i posiciones en el array

Punteros y arrays. Ejemplo 1

- Declara un array de enteros y un puntero.
- Luego asigna la dirección del primer elemento del array al puntero.
- Finalmente, utiliza un bucle for para recorrer el array utilizando el puntero y muestra el valor de cada elemento del array.

Punteros y cadenas de caracteres

- En C, una cadena de caracteres (`string`) es un array de caracteres terminado por el carácter ASCII 0 `('\0')`
- Definición de una cadena

```
char cad[5];
```

- **C no controla el acceso a posiciones fuera de la cadena**
- Manejo de la cadena utilizando el nombre del array

```
printf("%s",cad);
```

- Acceso a posiciones individuales, con sintaxis de array

```
printf("%c",cad[0]);
```

Punteros y cadenas de caracteres

- Inicialización
  - En la propia declaración:

```
char cad[5] = "hola";
char cad[5] = {‘h’,’o’,’l’,’a’,’\0’};
```

- **No está definido el operador de asignación**

```
cad = "hola";   /* ERROR */
```

- Utilizar strcpy para asignar un valor a una cadena

```
strcpy(cad,"hola“);
```

Punteros y cadenas de caracteres

- Algunas funciones útiles de la librería `<string.h>`

```
char *strcpy(char *dest, const char *src);
// Copia src en dest (sin controlar el tamaño de src)

size_t strlen(const char *str);
//Retorna la longitud de str (se supone terminado en \0)

char *strcat(char *dest, const char *src);
//Añade src a dest

int strcmp(const char *str1, const char *str2);
//Compara str1 y str2, devuelve 0 si son iguales;  menor que 0 si str1<str2 y mayor que 0 si str1>str2
```

Punteros y cadenas. Ejemplo

- Define tres cadenas, str1, str2 y str3.
- Inicializa str1 y str2.
- Copia una de esas cadenas en str3
- Concatena str1 y str2
- Compara str1 y str3

Arrays como parámetros en funciones

- Se pasan con sintaxis de punteros

```
void mifuncion(int *param);
```

```
void mostrar(int *elem, int n) {
   int i;
   for (i=0;i<n;i++)
      printf("%d",elem[i]);
}
```

```
int datos[10];
....
mostrar (datos,10);
```

Devolver Arrays en funciones?

- **No es posible**
- **Se pueden devolver punteros (cuidado con esto!)**
- Ejemplo:

```
char *strcpy(char *destination,const char*source);
```

Arrays y funciones

Si necesitamos pasar un array como parámetro de un función SIEMPRE se le pasa la dirección de comienzo de éste

Nunca se pasan por valor

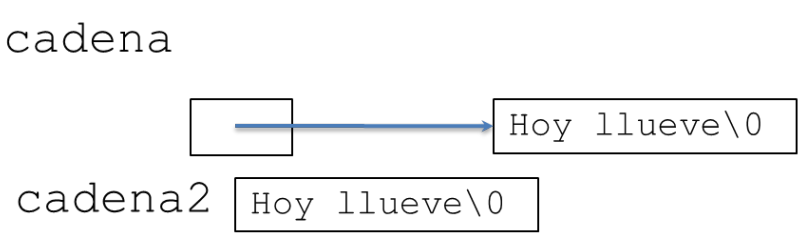
Arrays y funciones

```
void copiarCadena(char *destino, char *origen){
  int i = 0;
  while (origen[i]!='\0'){
    destino[i] = origen[i];
    i++;
  }
  destino[i]='\0';
}
```

Arrays y funciones

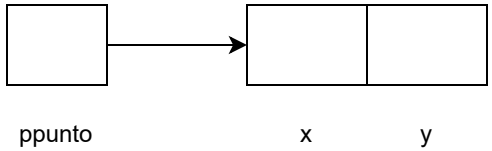
```
char *cadena = "hoy llueve" ; // puntero a constante
char cadena2[] = "hoy llueve" ; // array

char * ptr = cadena ;
```



Punteros y Registros (Structs)

- Los punteros pueden apuntar a cualquier tipo
  Ej:



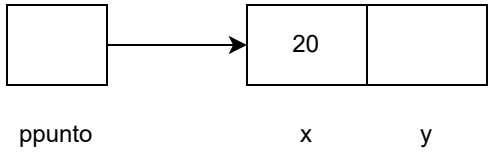

```
struct Punto *ppunto;
```

- Con typedef

```
typedef struct Punto *Ppunto;
Ppunto ppunto;
```

Punteros y Registros (structs)

- El acceso a los miembros puede hacerse de dos formas
- Sintaxis 1:




```
(*ppunto).x=20;
```

- Sintaxis 2:

```
ppunto->x=20;
```

Punteros a funciones

- En C, el nombre de una función es un puntero a dicha función

```
int esMayorInt (int *a, int *b) { return *a > *b ; }

int esMayorDouble (double *a, double *b) { return *a > *b ; }

int esMayor(void *a, void *b, int (*f)(void *, void *)) {
  return (*f)(a, b) ;
}
```