

Auxiliar 3 - Strings

Otoño 2025

Profesores: Alexandra Ibarra

Luis Mateu

Rodrigo Urrea

Auxiliares: Luciano Márquez

Tomás Vergara

Resumen

Punteros

- Apuntan a un lugar de la memoria.
- Se puede asignar, evaluar, y declarar.
- El operador contenido `*` se usa para acceder al valor.
- El operador dirección `&` se usa para acceder al lugar de memoria.

```
int val = 3;
int* p; // declarar
p = &val; // asignar
int* q = &val;
printf("%d\n", q == p) // evaluar
```

Areglos

- Conjunto numerado del mismo tipo.
- Se declaran con el operador de subíndice `[]`, que también se usa para acceder a los contenidos del arreglo

```
int a[10]; // arreglo de 10 ints
a[2] = 3; // el terecer elemento del
arreglo es 3
printf("%d\n", a[10]); // error!
```

- No es exactamente lo mismo que un puntero.
- Un arreglo representa un sección contigua de memoria.

```
int *p;
int q[1];
q[0] = 2; // :D
*p = 2; // error!
```

Aritmetica

- Se pueden usar operaciones aritméticas sobre punteros.
- Viene de la mano con a donde apunta un puntero.
- Al declarar un arreglo cualquiera `p[10]`, `p` es un puntero que apunta al primer valor del arreglo.
- Sumando o restando enteros se puede avanzar o retroceder en el arreglo.

$\&\text{dir}[\text{ind}] \iff \text{dir} + \text{ind};$

Es decir,

$\text{dir}[\text{ind}] = x \iff *(\text{dir} + \text{ind}) = x;$

Strings

- Arreglo de caracteres que termina en 0.
- Al igual que un arreglo, uno puede acceder a cada carácter.

```
char str[] = {'H', 'o', 'l', 'a', 0};
char *r = str;
printf("%c", *(r+3)); // imprime 'a'
```

- Se pueden crear string inmutables y mutables.

```
char *str2 = "Hola"; // Inmutable
*str2 = 'h'; // Segmentation fault
char str3[] = "Hola"; // Mutable
str3 = 'h'; // str3 referencia a
"hoLa"
```

Funciones relevantes

- **int strlen(char* s):** entrega el tamaño del string. No considera el 0 al final del string por lo que entrega el tamaño real del string.
`int len = strlen("Hola"); // l = 4`
- **char *strcpy:** copia el string s en el string d, d debe tener tamaño mayor o igual a strlen(s) 1.
`char d[20];
strcpy(d, "Hola");
printf("%s", d); // Imprime "Hola"`
- **int strcmp(char *s, char *r):** compara lexicográficamente los strings s y r.
 - `strcmp(s, r) < 0` $\iff s < r$
 - `strcmp(s, r) = 0` $\iff s = r$
 - `strcmp(s, r) > 0` $\iff s > r$

Preguntas

P1. Programe las siguientes funciones

```
void to_lower(char* s);  
void to_upper(char* s);
```

Estas funciones reciben un string s. La primera convierte todas las letras mayúsculas del string en minúsculas y la segunda hace lo contrario.

P2. Escriba una función que retorne 1 si el string s es un palíndromo y 0 en el caso contrario.

```
int palindromo(char* s)
```

P3. Escriba una función que reciba un string s y lo invierta en el lugar, es decir, usando memoria adicional $O(1)$.

```
void reverse(char* s)
```

P4. Escriba una función que retorne el caracter de s que tiene el mayor número de repeticiones en el string. Si hay varios con la mayor cantidad, retorna cualquiera.

```
char mas_repetido(char* s)
```