C-String

Ejemplos básicos

C-String

Es una convención que dicta que un arreglo de caracteres representa un texto que **finaliza con el primer carácter nulo (`\0`)** que se encuentre

Este carácter nulo es crucial, ya que todas las funciones de la biblioteca estándar de C (`<string.h>`) dependen de él para saber dónde termina la cadena.

Características principales:

- **Terminador nulo (`\0`):** Es obligatorio. Indica el final de la cadena, no necesariamente el final del arreglo.
- **Tamaño del arreglo:** Debe ser, como mínimo, la longitud del texto más uno (para el `\0`).
- **Compatibilidad con la biblioteca estándar:** Puedes usar funciones como `strlen()`, `strcpy()`, `printf("%s", ...)` y otras.

Declaración e inicialización:

a) Inicialización con un literal de cadena: El compilador se encarga de añadir el `\0` automáticamente. Es la forma más fácil y segura.

```
char saludo[] = "Hola";
// El compilador reserva 5 bytes: 'H','o','l','a','\0'
```

Declaración e inicialización:

b) Inicialización manual: Debes recordar añadir el `\0` explícitamente.

```
char palabra[5];
palabra[0] = 'H';
palabra[1] = 'o';
palabra[2] = 'l';
palabra[3] = 'a';
palabra[4] = '\0'; // ¡Fundamental!
```

Declaración e inicialización:

c) Puntero a un literal de cadena: Aquí, `mensaje` es un puntero que apunta a una zona de memoria (generalmente de solo lectura) donde se almacena "Texto inmutable". No debes intentar modificar su contenido.

```
const char *mensaje = "Texto inmutable";
```

```c
#include <stdio.h>
#include <string.h> // Necesario para las funciones de strings
int main() {
    char nombre[20] = "Juan"; // Deja espacio extra para modificaciones
    char apellido[] = " Pérez";
    // 1. Calcular la longitud (no incluye el \0)
    printf("Longitud del nombre: %zu\n", strlen(nombre)); // Salida: 4
    // 2. Concatenar (añadir) una cadena a otra
    // strncat es más seguro que strcat porque limita los bytes a añadir
    strncat(nombre, apellido, sizeof(nombre) - strlen(nombre) - 1);
    // 3. Imprimir la cadena completa
    printf("Nombre completo: %s\n", nombre); // Salida: Juan Pérez
    // 4. Comparar cadenas
    if (strcmp(nombre, "Juan Pérez") == 0) {
        printf("Los nombres son iguales.\n");
    }
    return 0;
}
```

Manejo común de los c-string

Estos ejemplos prácticos que cubren las operaciones más comunes de manejo y manipulación de C-strings.

1. Declaración, Inicialización y Longitud

Este ejemplo muestra cómo declarar una cadena, obtener su longitud real y comparar esa longitud con el tamaño total del búfer de memoria asignado.

- **Funciones clave:** `strlen()`, `sizeof()`

```c
#include <stdio.h>
#include <string.h>

int main() {
    // El arreglo tiene espacio para 19 caracteres + el terminador nulo '\0'
    char mi_ciudad[20] = "Rosario";

    // strlen() cuenta los caracteres HASTA encontrar el '\0'.
    size_t longitud = strlen(mi_ciudad);

    // sizeof() devuelve el tamaño total en bytes del arreglo, sin importar su contenido.
    size_t tamano_buffer = sizeof(mi_ciudad);

    printf("Cadena: '%s'\n", mi_ciudad);
    printf("La longitud de la cadena (strlen) es: %zu\n", longitud);         // Salida: 7
    printf("El tamaño del búfer (sizeof) es: %zu bytes\n", tamano_buffer); // Salida: 20

    return 0;
}
```

Explicación

Es fundamental entender la diferencia entre `strlen()` y `sizeof()`. `strlen()` recorre la cadena en tiempo de ejecución para contar caracteres, mientras que `sizeof()` es un operador que determina el tamaño de la memoria reservada en tiempo de compilación.

2. Leer una Cadena de Forma Segura

Leer texto del usuario es una de las operaciones más peligrosas si no se hace con cuidado. El método recomendado es usar `fgets()`, que previene desbordamientos de búfer (buffer overflows).

- **Función clave:** `fgets()`

```
#include <stdio.h>
#include <string.h>
int main() {
    char nombre_usuario[50]; // Búfer de 50 bytes
    printf("Por favor, introduce tu nombre completo: ");
    // fgets lee hasta un máximo de sizeof(nombre_usuario) - 1 caracteres.
    // Siempre añade un '\0' y es seguro contra desbordamientos.
    // Lee desde la entrada estándar (stdin).
    if (fgets(nombre_usuario, sizeof(nombre_usuario), stdin) != NULL) {
        // Un problema común: fgets() también almacena el salto de línea (\n) si hay espacio.
        // Lo buscamos y lo eliminamos.
        size_t len = strlen(nombre_usuario);
        if (len > 0 && nombre_usuario[len - 1] == '\n') {
            nombre_usuario[len - 1] = '\0'; // Lo reemplazamos con el terminador nulo
        }
        printf("¡Hola, %s! Bienvenido.\n", nombre_usuario);
    }
    return 0;
}
```

Explicación

`fgets()` es superior a funciones como `gets()` (obsoleta y extremadamente peligrosa) o `scanf("%s", ...)` porque te obliga a especificar el tamaño máximo del búfer. Esto garantiza que nunca se escribirá más allá de los límites del arreglo.

3. Copiar y Concatenar Cadenas

Manipular cadenas a menudo implica copiarlas a un nuevo búfer o unirlas. Para esto, siempre debes usar las versiones "n" de las funciones (`strncpy`, `strncat`) para controlar el tamaño y evitar desbordamientos.

- **Funciones clave:** `strncpy()`, `strncat()`

```c
#include <stdio.h>
#include <string.h>

int main() {
    char saludo[30] = "Hola"; // Búfer con suficiente espacio
    char nombre[] = "Mundo";

    // Concatenar de forma segura
    strncat(saludo, ", ", sizeof(saludo) - strlen(saludo) - 1);
    strncat(saludo, nombre, sizeof(saludo) - strlen(saludo) - 1);
    printf("Cadena concatenada: '%s'\n", saludo); // Salida: 'Hola, Mundo'

    // Copiar de forma segura
    char destino[20];
    strncpy(destino, saludo, sizeof(destino) - 1);

    // ¡MUY IMPORTANTE! strncpy no garantiza el terminador nulo
    // si la fuente es más grande.
    // Hay que asegurarlo manualmente.
    destino[sizeof(destino) - 1] = '\0';
    printf("Cadena copiada: '%s'\n", destino); // Salida: 'Hola, Mundo'

    return 0;
}
```

4. Búsqueda de Caracteres y Subcadenas

Puedes buscar la primera aparición de un carácter específico o de una secuencia de caracteres (subcadena) dentro de otra cadena.

- **Funciones clave:** `strchr()`, `strstr()`

```c
#include <stdio.h>
#include <string.h>
int main() {
    char frase[] = "El lenguaje C es un lenguaje potente.";
    char caracter_a_buscar = 'j';
    char subcadena_a_buscar[] = "lenguaje";
    // Buscar un carácter
    char* resultado_char = strchr(frase, caracter_a_buscar);
    if (resultado_char != NULL) {
        printf("Carácter '%c' encontrado en la posición: %ld\n", caracter_a_buscar, resultado_char - frase);
    } else {
        printf("Carácter '%c' no encontrado.\n", caracter_a_buscar);
    }
    // Buscar una subcadena
    char* resultado_sub = strstr(frase, subcadena_a_buscar);
    if (resultado_sub != NULL) {
        printf("Subcadena '%s' encontrada en la posición: %ld\n", subcadena_a_buscar, resultado_sub - frase);
        printf("El resto de la cadena desde ahí es: '%s'\n", resultado_sub);
    } else {
        printf("Subcadena '%s' no encontrada.\n", subcadena_a_buscar);
    }
    return 0;
}
```

Explicación

Estas funciones devuelven un **puntero** a la ubicación donde encontraron la coincidencia, o `NULL` si no la encontraron. Restando el puntero original del puntero del resultado (`resultado_char - frase`) puedes obtener el índice numérico de la posición.

5. Tokenización (Dividir una Cadena en Partes)

Una tarea muy común es dividir una cadena en "tokens" o partes, basándose en un delimitador (como un espacio, una coma, etc.). Esto es útil para procesar comandos o datos formateados.

- **Función clave:** `strtok()`

Advertencia: `strtok()` es una función destructiva, ya que modifica la cadena original insertando caracteres nulos (`\0`) donde encuentra los delimitadores.

```c
#include <stdio.h>
#include <string.h>

int main() {
    // OJO: Usamos un arreglo, no un puntero a un literal, porque strtok() lo modificará.
    char csv_data[] = "Juan,Perez,42,Rosario";
    const char delimitador[] = ",";

    // La primera llamada a strtok() requiere la cadena.
    char* token = strtok(csv_data, delimitador);
    printf("Datos desglosados:\n");
    while (token != NULL) {
        printf(" - %s\n", token);

        // Las llamadas siguientes usan NULL para continuar desde donde se quedó.
        token = strtok(NULL, delimitador);
    }

    return 0;
}
```

Explicación

En la primera llamada, `strtok()` encuentra la primera coma y la reemplaza por un `\0`, devolviendo un puntero al inicio ("Juan"). En las llamadas siguientes, al pasarle `NULL`, le indicas que continúe trabajando sobre la misma cadena original desde la posición donde se quedó.