Principios de Programación

Cadenas/Bool

Texto en C con arreglos de `char` y un cierre breve sobre `bool`: terminador nulo, declaración, lectura básica, recorrido y operaciones elementales.

Recordando la semana anterior

Base previa

- Ya se trabajó con arreglos unidimensionales
- Ya se accedió a posiciones por índice
- Ya se usaron matrices para agrupar varios arreglos

Necesidad de representar texto

SIN CADENAS

```c
char letra1 = 'H';
char letra2 = 'o';
char letra3 = 'l';
char letra4 = 'a';

printf("%c%c%c%c\n",
       letra1, letra2, letra3, letra4);
```

Problema

- Cada carácter queda en una variable distinta
- Recorrer, copiar o comparar el texto se vuelve incómodo
- Si cambia la longitud, el código debe reescribirse

Pregunta

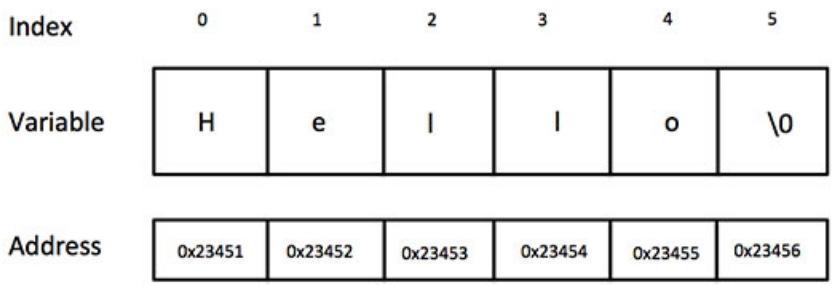
Si un programa necesita tratar varias letras como una sola unidad, conviene agruparlas en un arreglo de `char`.

Definición

Una cadena es una secuencia de caracteres almacenados en posiciones consecutivas de un arreglo de `char` y finalizada por el carácter nulo `'\0'`.

Lectura

- Cada letra ocupa una posición del arreglo
- El texto útil termina cuando aparece `'\0'`
- El arreglo puede tener más espacio que el texto visible



Representación de una cadena en memoria

Cadena `saludo`

El terminador nulo no se imprime, pero marca el final lógico del texto.

Índice 0	Índice 1	Índice 2	Índice 3	Índice 4
`saludo[0]`	`saludo[1]`	`saludo[2]`	`saludo[3]`	`saludo[4]`
'H'	**'o'**	**'l'**	**'a'**	**'\0'**

Posiciones

La palabra visible tiene 4 letras.

Tamaño mínimo

El arreglo necesita 5 posiciones.

Regla

Si el texto tiene `n` caracteres, se necesita espacio para `n + 1`.

Declaración de una cadena

SINTAXIS

```
char nombre[tamano];
```

EJEMPLOS

```
char nombre[20];
char cedula[9];
char mensaje[80];
```

Componentes

- `char`: tipo de cada elemento
- `nombre`: identificador del arreglo
- `tamano`: cantidad de posiciones disponibles

Regla

Si se quiere guardar una cadena de hasta `19` caracteres visibles, el arreglo debe tener al menos `20` posiciones para dejar lugar a `'\0'`.

Inicialización de cadenas

CON COMILLAS

```
char saludo[5] = "Hola";
char pais[] = "Uruguay";
```

CON LISTA DE CARACTERES

```
char sigla[4] = {'U', 'T', 'U', '\0'};
char ok[3] = {'s', 'i', '\0'};
```

Lectura

- `"Hola"` incluye el terminador nulo al inicializar
- `char pais[] = "Uruguay";` deja que el compilador calcule el tamaño

Cuidado

Un `char` guarda un solo carácter. Una cadena necesita varias posiciones consecutivas del arreglo.

Lectura y escritura de una palabra

ENTRADA Y SALIDA

```c
#include <stdio.h>

int main() {
    char nombre[20];

    printf("Ingrese su nombre: ");
    scanf("%19s", nombre);

    printf("Hola, %s\n", nombre);
    return 0;
}
```

POSIBLE EJECUCIÓN

```
Ingrese su nombre: Ana
Hola, Ana
```

Lectura

- Para `%s`, se pasa el nombre del arreglo sin `&`
- `%19s` deja una posición libre para `'\0'`
- El mismo `%s` sirve para imprimir la cadena

Límite de `%s`

EJEMPLO

```c
char ciudad[20];

scanf("%19s", ciudad);
printf("[%s]\n", ciudad);
```

SI EL USUARIO ESCRIBE

```
Montevideo Este
```

SALIDA

```
[Montevideo]
```

Interpretación

En esta forma básica, `%s` sirve para una palabra sin espacios. Si el problema requiere una frase completa, hay que usar otra estrategia de lectura.

Recorrer una cadena con `for`

RECORRIDO

```c
char palabra[] = "casa";
int i;

for (i = 0; palabra[i] != '\0'; i++) {
    printf("%c\n", palabra[i]);
}
```

Condicion

- El índice empieza en `0`
- El recorrido continúa mientras no aparezca `'\0'`
- No hace falta conocer antes la longitud visible

Ejemplo: contar vocales

PROCESAMIENTO SOBRE LA CADENA

```c
char palabra[] = "murcielago";
int i, vocales = 0;

for (i = 0; palabra[i] != '\0'; i++) {
    if (palabra[i] == 'a' || palabra[i] == 'e' ||
        palabra[i] == 'i' || palabra[i] == 'o' ||
        palabra[i] == 'u') {
        vocales++;
    }
}

printf("Vocales: %d\n", vocales);
```

Lectura

- Una cadena puede procesarse como cualquier otro arreglo
- Cada paso examina un carácter puntual
- La lógica termina al llegar al terminador nulo

RESULTADO

```
Vocales: 5
```

OPERACIONES MÁS USADAS

Función	Qué hace	Ejemplo breve
`strlen(cad)`	Devuelve la longitud visible.	`strlen("hola") -> 4`
`strcpy(dest, orig)`	Copia una cadena en otra.	`strcpy(nombre, "Ana")`
`strcmp(a, b)`	Compara dos cadenas.	`strcmp(clave, "admin")`
`strcat(a, b)`	Concatena `b` al final de `a`.	`strcat(saludo, "!")`

Regla

Estas funciones trabajan sobre arreglos de `char` bien formados. La cadena debe terminar en `'\0'` y el arreglo de destino debe tener espacio suficiente.

Ejemplo con `strlen()`

POR QUÉ IMPORTA \\0

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

int main() {
    char texto[] = "hola\0mundo";

    printf("%s\n", texto);
    printf("Longitud: %zu\n", strlen(texto));
    return 0;
}
```

SALIDA

```
hola
Longitud: 4
```

Lectura

`strlen()` cuenta hasta el primer `'\0'`. Aunque el arreglo tenga más caracteres después, la cadena lógica ya terminó.

Comparar cadenas

COMPARACIÓN INCORRECTA

```c
if (clave == "admin") {
    printf("Acceso permitido\n");
}
```

COMPARACIÓN CORRECTA

```c
if (strcmp(clave, "admin") == 0) {
    printf("Acceso permitido\n");
}
```

Lectura

== no sirve para este caso. Para cadenas, en este curso se compara el contenido con `strcmp`.

Resultado de `strcmp`

- `0` si son iguales
- Menor que `0` si la primera es menor
- Mayor que `0` si la primera es mayor

Reservar espacio sin considerar `\\0`

CÓDIGO CON ERROR

```
char saludo[4] = "Hola";
```

MENSAJE / PROBLEMA

El texto visible `"Hola"` tiene 4 caracteres, pero la cadena necesita una posición adicional para el terminador nulo. Ese arreglo no alcanza para almacenar la cadena correctamente.

CORRECCIÓN

Corregir de una de estas formas:

- `char saludo[5] = "Hola";`
- `char saludo[] = "Hola";`

En ambos casos queda espacio para `'\0'`.

La condición puede fallar aunque el texto coincida

CÓDIGO CON ERROR

```
char clave[10];

scanf("%9s", clave);

if (clave == "admin") {
    printf("OK\n");
}
```

MENSAJE / PROBLEMA

La comparación no verifica el contenido de la cadena de la forma esperada. En este caso corresponde usar una función de `string.h`.

CORRECCIÓN

Incluir `#include <string.h>` y comparar con:

```
if (strcmp(clave, "admin") == 0) {
    printf("OK\n");
}
```

Ejercicio 1

Leer una palabra y mostrar:

- su longitud
- cuántas vocales tiene
- su última letra

Ejercicio 2

Leer dos cadenas e indicar:

- si son iguales
- cuál es más larga
- cuál aparece primero alfabéticamente

Ejercicio 3

Cargar 5 nombres en una matriz de cadenas y luego mostrarlos en orden inverso al de ingreso.

CON `int`

```c
int activo = 1;
int error = 0;
```

CON `bool`

```c
#include <stdbool.h>

bool activo = true;
bool error = false;
```

Lectura

- Para el uso básico del curso, `bool` representa la misma idea lógica de `0` y `1`
- `false` equivale a `0`
- `true` equivale a `1`

Idea

`stdbool.h` no cambia la lógica de las condiciones. Aporta nombres más claros para escribir verdadero y falso.

Qué define `stdbool.h`

RECORTE STDBOOL.H

```
...
#define bool  bool
#define true  1
#define false 0
...
```

Lectura

- `bool` se apoya en `_Bool`
- `true` está definido como `1`
- `false` está definido como `0`

Ejemplos con `bool`

USO BÁSICO

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

int main() {
    int edad = 17;
    bool mayor = edad >= 18;
    bool listo = true;

    printf("%d\n", mayor);
    printf("%d\n", listo);

    return 0;
}
```

SALIDA POSIBLE

```
0
1
```

Lectura

- Una expresión lógica puede guardarse directamente en un `bool`
- Al imprimirlo con `%d`, se ve `0` o `1`
- En un `if`, `bool` se usa igual que cualquier condición del curso