Programación I

Pizarras y Ejemplos de clase, noviembre 2025 (II)

Este documento contiene las pizarras y ejemplos de código que se utilizaron en las clases de Programación I del grupo G1M3, durante la segunda mitad el mes de noviembre de 2025, dedicadas a las cadenas de caracteres.

1 . Clase 19 noviembre 2025: Cadenas de caracteres (I)

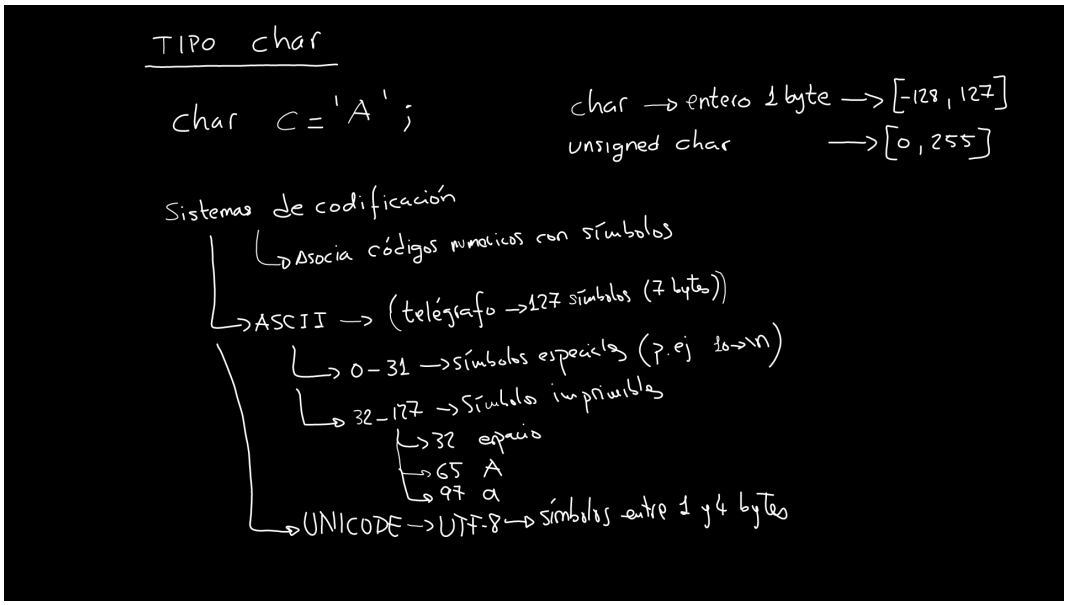

Figura 1: El tipo *char* en C y los sistemas de codificación.

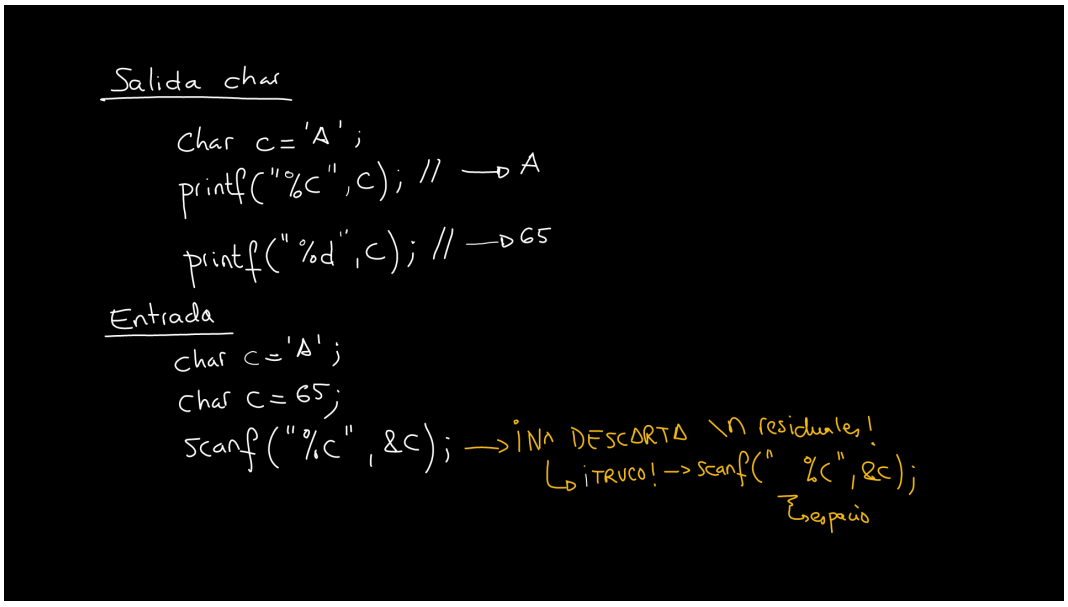

Figura 2: Entrada y salida de caracteres individuales.

Al contrario de lo que sucede con otros especificadores de formato, el especificador %c para leer caracteres no descarta los caracteres fin de línea '\n' que pudieran quedar en el búfer de la entrada estándar. Para que los descarte, hay que poner un espacio antes del especificador, como se hace en el Ejemplo 1.

Ejemplo 1 Entrada estándar de caracteres individuales

```
#include <stdio.h>

int main() {

   // Salida de un carácter, como carácter y como código ASCII
   char c = 'A';
   printf("%c %d\n", c, c);

   // Inicialización de un carácter a partir de su código ASCII
   c = 97;
   printf("%c %d\n", c, c);

   // El formato %lf o el formato %d descartan los \n residuales que pudiera haber
   double x;
   printf("Teclea double: ");
   scanf("%lf", &x);

   // El formato %c, en cambio, no descarta los \n residuales
   printf("Teclea char: ");
   scanf("%c", &c);
   printf("-%c- -%d-\n", c, c);

   // Para descartar los \n residuales, hay que poner un espacio antes del especificador de formato
   printf("Teclea char: ");
   scanf(" %c", &c);
   printf("-%c- -%d-\n", c, c);

   return 0;
}
```

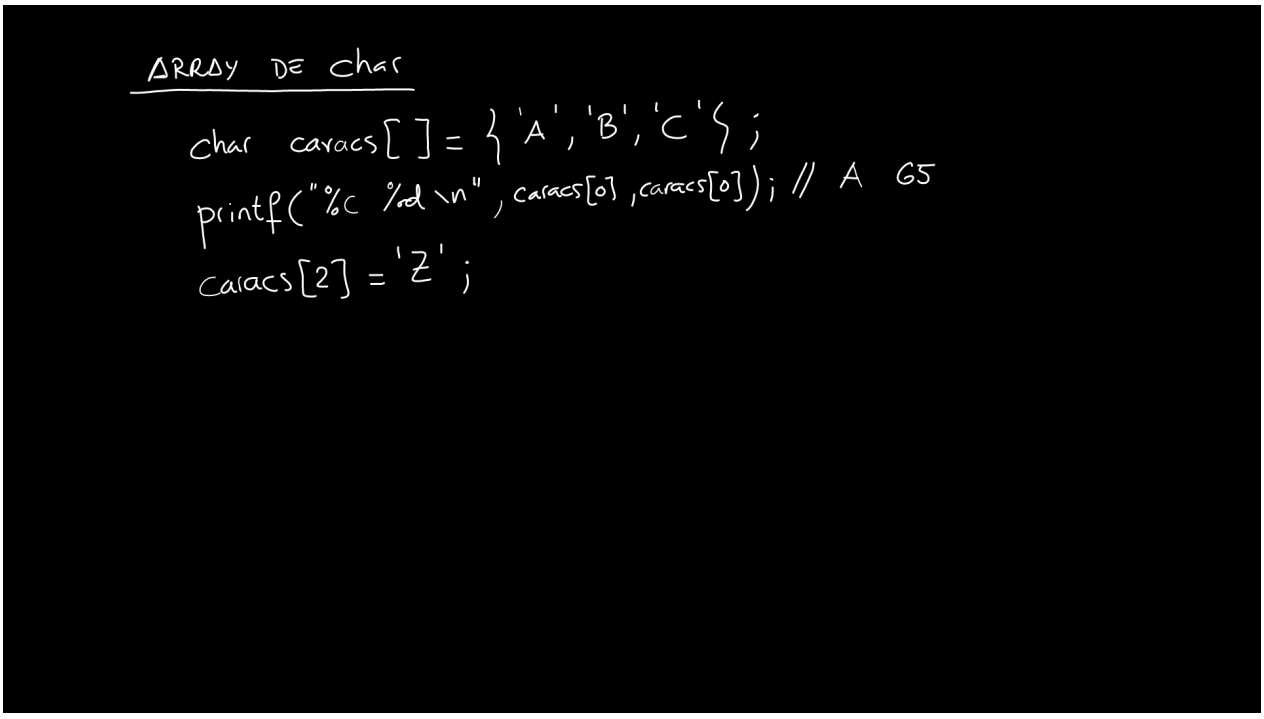

Figura 3: Array cuyos elementos son caracteres individuales.

Un array puede tener como elementos caracteres individuales. Pero, si no cumple que el último carácter sea el '\0', no es una *cadena de caracteres*. En el Ejemplo 2, se declara un array de caracteres individuales, se modifica uno de sus elementos y se imprimen los elementos del array, como caracteres y como códigos ASCII.

Ejemplo 2 Arrays de caracteres individuales

```
#include <stdio.h>

int main() {

   char caracs[] = {'A', 'B', 'C'};

   printf("%c %d\n", caracs[0], caracs[0]);

   caracs[2] = 'Z';
   printf("%c %d\n", caracs[2], caracs[2]);

   for(int i=0; i<3; i++) {
      printf("%c %d\n", caracs[i], caracs[i]);
   }
   return 0;
}
```

¡Atención!

En este curso, solo usaremos caracteres ASCII. Se excluyen, por tanto, las letras acentuadas, la letra ñ y otros caracteres del español.

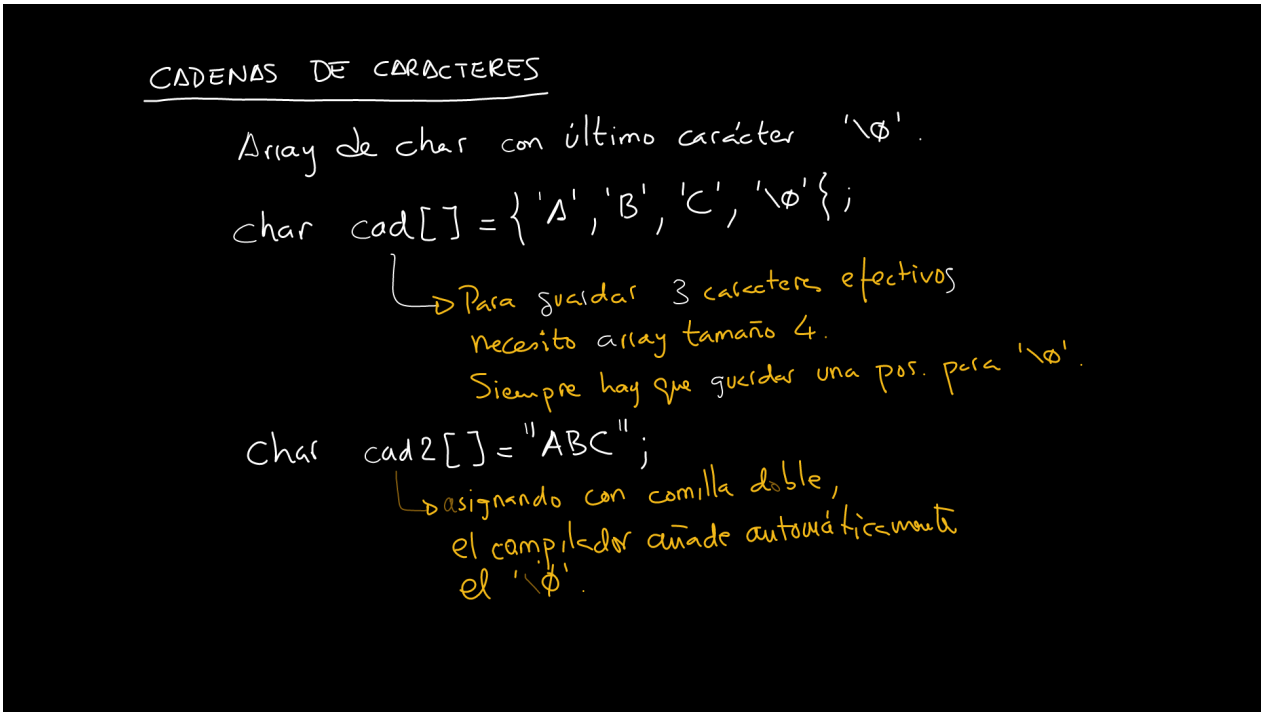

Figura 4: Cadenas de caracteres: arrays de *char* acabados en '\0'.

Una *cadena de caracteres* es un array con elementos del tipo *char*, pero en el que el último carácter de la cadena es el carácter '\0'. Podemos usar dos formas de inicialización:

- Carácter a carácter en un array. En este caso, tenemos que añadir como último carácter el '\0'.
- Entre comillas dobles. En este caso, el compilador añade automáticamente el carácter '\0'.

En cualquiera de los casos, para guardar una cadena que pueda albergar *n* caracteres, hará falta un array de longitud *n+1*: *n* caracteres más el carácter '\0'.

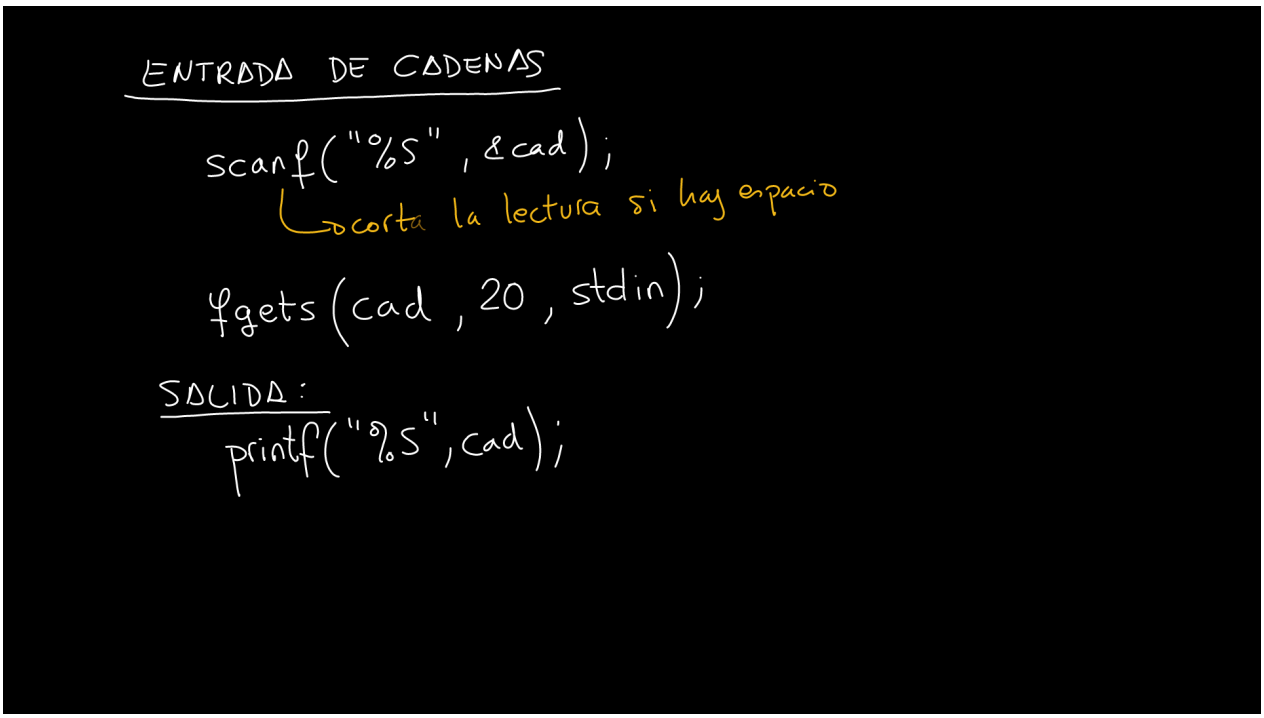

Figura 5: Entrada y salida de cadenas de caracteres.

Cuando leemos cadenas de la entrada estándar utilizando la función *fgets()*, el último carácter leído será el '\n'. El Ejemplo 3 muestra esta situación imprimiendo los caracteres leídos y los códigos ASCII de dichos caracteres. El código aSCII 10 corresponde al '\n'.

Ejemplo 3 \n residual al leer cadenas con fgets()

```
#include <stdio.h>

int main() {

   char nombre[8];

   printf("Nombre: ");
   fgets(nombre, 8, stdin);

   //printf("%s %s\n", nombre, nombre);

   for(int i=0; i<8; i++) {
      printf("%c ", nombre[i]);
   }
   printf("----------------------\n");

   for(int i=0; i<8; i++) {
      printf("%d ", nombre[i]);
   }
   printf("----------------------\n");

   return 0;
}
```

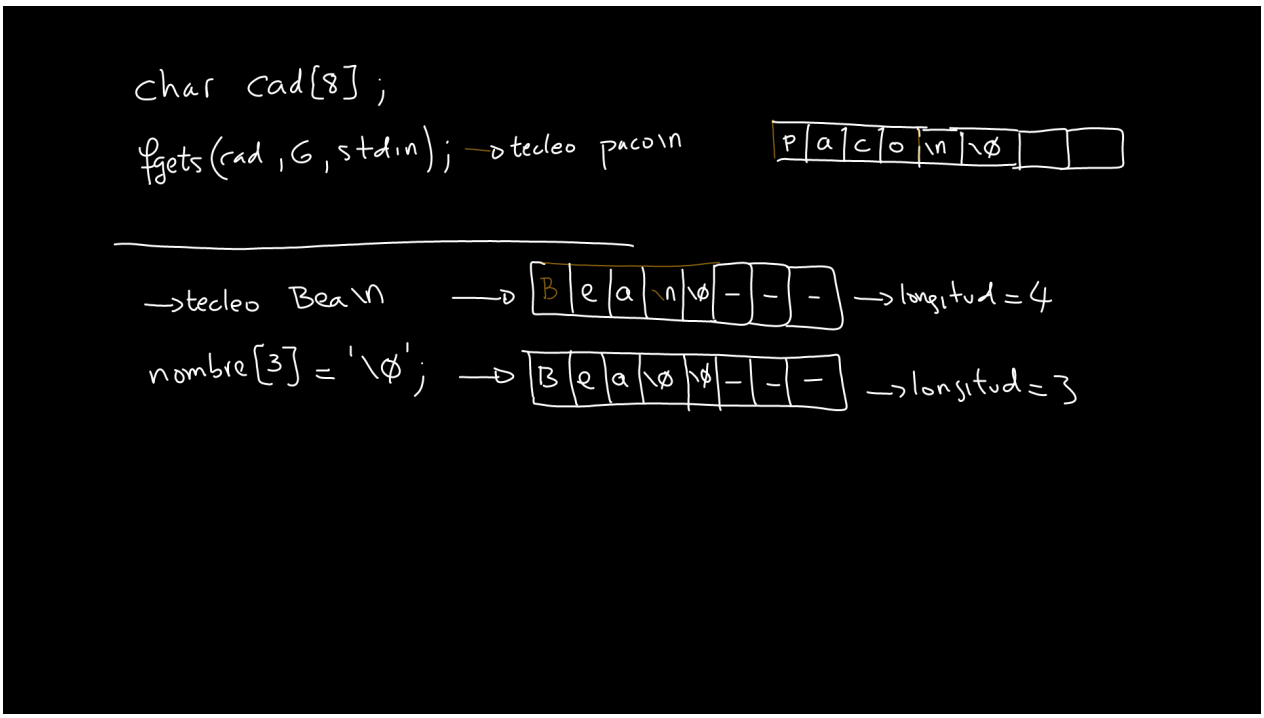

Figura 6: La cadena de caracteres puede que no ocupe todas las posiciones disponibles del array.

2 . Clase 21 noviembre 2025: cadenas de caracteres (II)

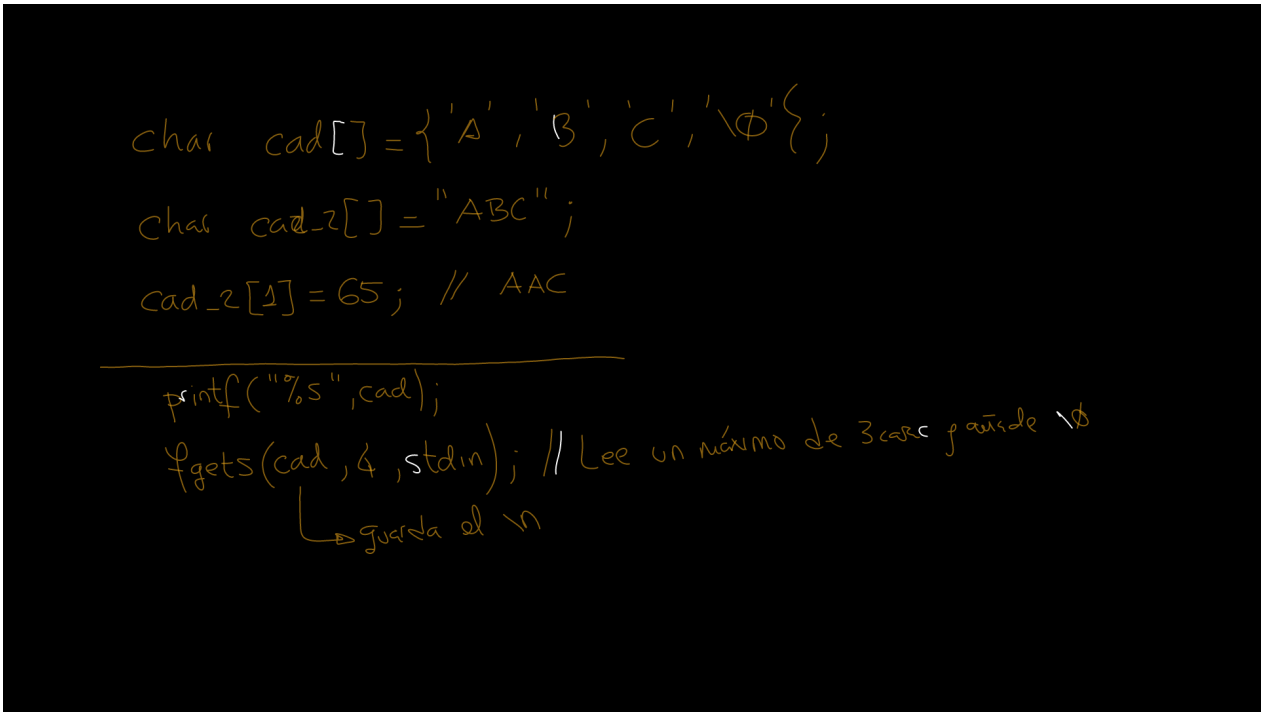

Figura 7: Distintas maneras de inicializar una cadena de caracteres. Salidas con *printf()* y entradas con *fgets()*.

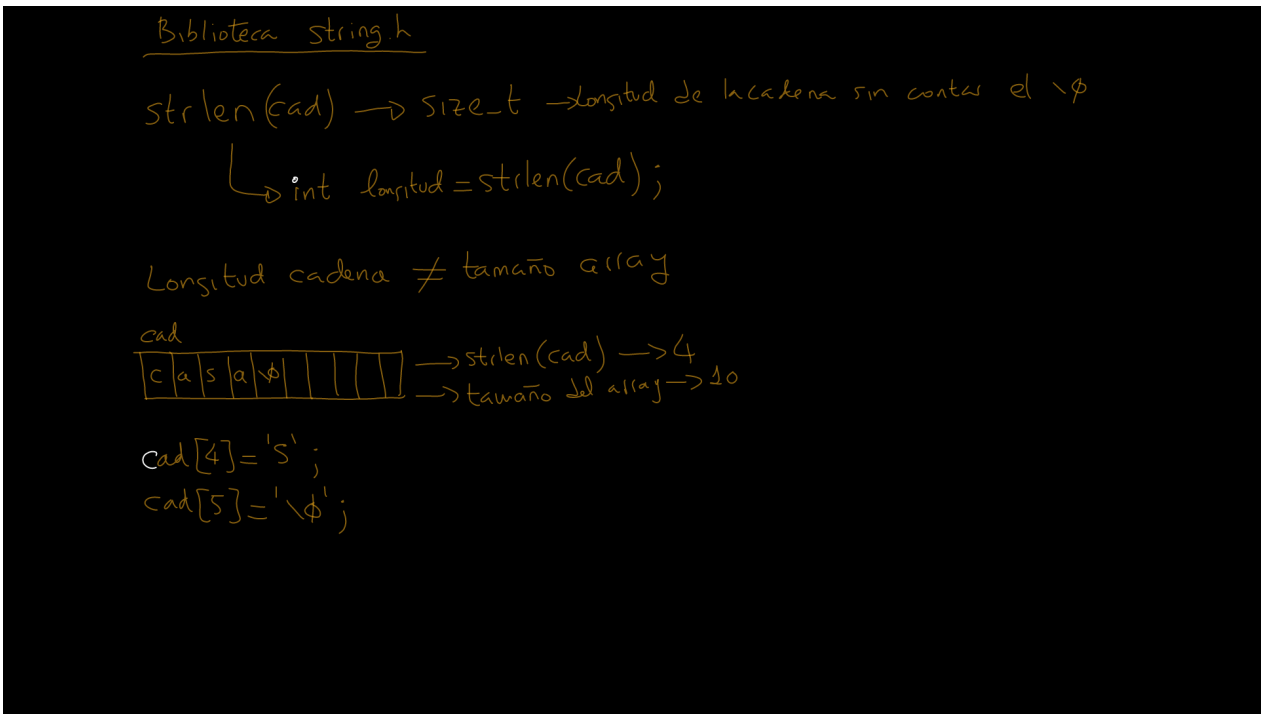

Figura 8: La función *strlen()* de la biblioteca *string.h*.

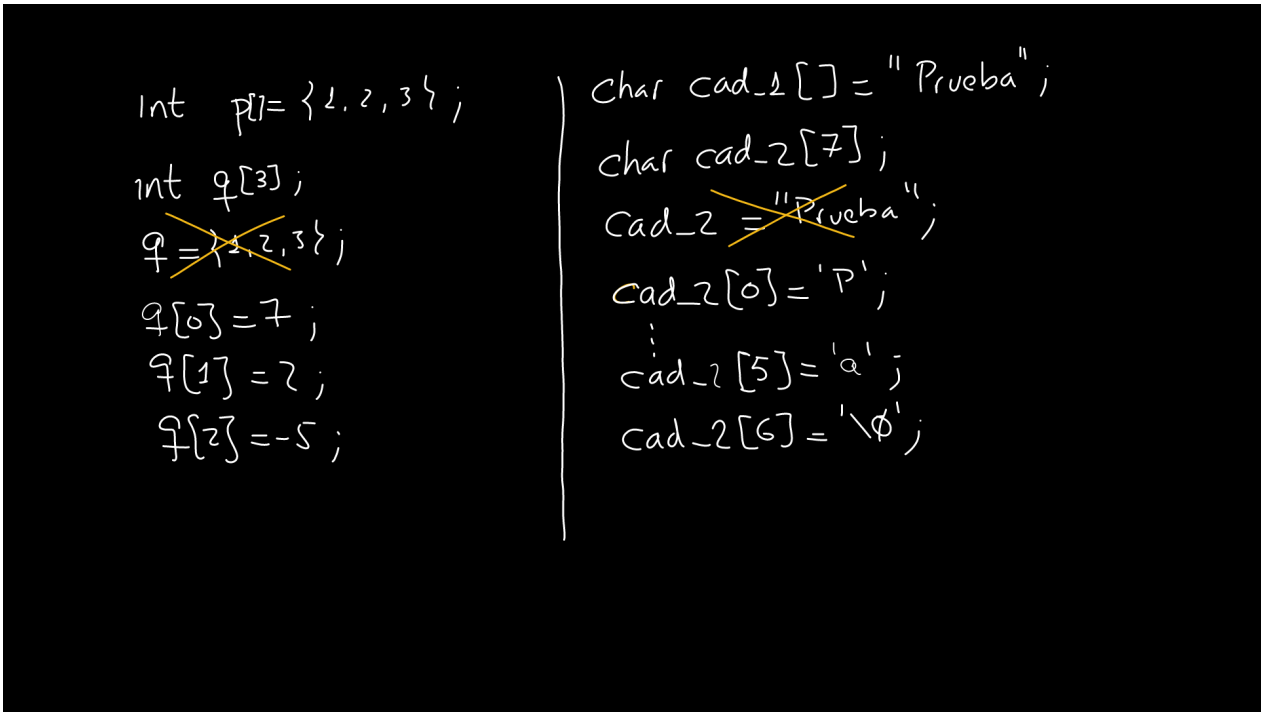

Figura 9: Ningún array se puede inicializar dos veces. La asignación directa del array solo es posible en la misma instrucción de declaración.

Página para consultar la biblioteca *string.h*

Puedes consultar la funciones de la biblioteca *string.h* en el siguiente enlace:

`https://www.w3schools.com/c/c_ref_string.php`

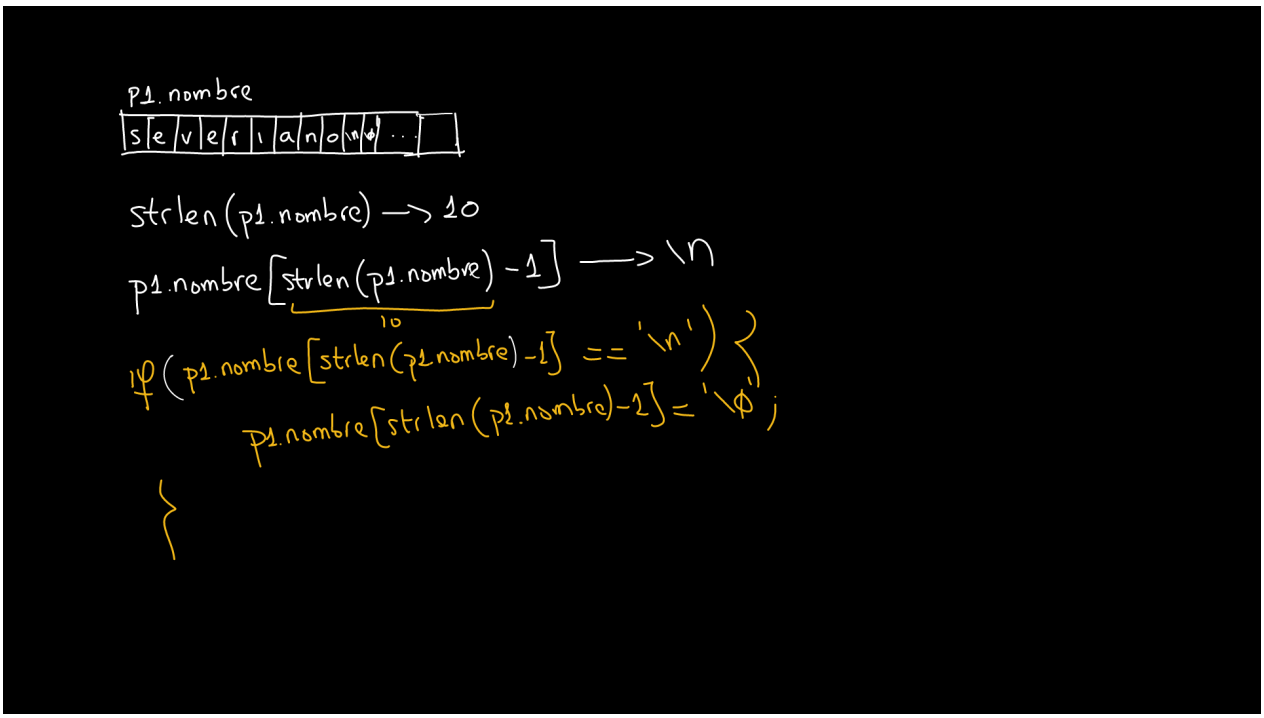

Figura 10: Eliminar el carácter '\n' residual en cadenas leídas desde la entrada estándar (método de sustitución del último carácter de la cadena).

El Ejemplo 4 muestra cómo eliminar el '\n' residual, sustituyendo el último carácter de la cadena por '\0'. Es necesario utilizar la función *strlen()* de la biblioteca *string.h* para saber el índice del último carácter de la cadena.

Ejemplo 4 Sustituyendo el '\n'

```
 por '\textbackslash 0'.
#include <stdio.h>
#include <string.h>

int main() {

   char nombre[8];

   printf("Nombre: ");
   fgets(nombre, 8, stdin);

   //printf("%s %s\n", nombre, nombre);

   int longitud = strlen(nombre); // La longitud incluye el \n
   printf("%s\n", nombre);
   printf("Longitud: %d\n", longitud);

   if(nombre[longitud-1] == '\n') {
      nombre[longitud-1] = '\0';
   }

   longitud = strlen(nombre); // Ahora la longitud ya no incluye el \n
   printf("%s\n", nombre);
   printf("Longitud: %d\n", longitud);

   return 0;
}
```

El Ejemplo 5 muestra un programa que utiliza una estructura en la que los campos son cadenas de caracteres.

Ejemplo 5 Ejemplo de estructura con campos cadenas de caracteres

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

typedef struct {
   char nombre[20];
   char apellidos[40];
} Persona;

int main() {
   Persona p1 = {"Pedro", "Moreno"}; // Inicialización directa de los campos cadena de caracteres

   printf("%s, %s\n", p1.apellidos, p1.nombre);

   return 0;
}
```

Para asignar valor a campos de cadenas de manera dinámica (en tiempo de ejecución) se pueden utilizar funciones de *string.h*, como se hace en el Ejemplo 6.

Ejemplo 6 Asignación dinámica de cadenas de caracteres.

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

typedef struct {
   char nombre[20];
   char apellidos[40];
} Persona;

int main() {
   Persona p1;
   //p1.nombre = "Elena";     // ¡ERROR, no se puede inicializar dos veces!
   //p1.apellidos = "Morena"; // ¡ERROR, no se puede inicializar dos veces!

   strcpy(p1.nombre, "Elena");     // Asignación dinámica
   strcpy(p1.apellidos, "Morena"); // Asignación dinámica

   printf("%s, %s\n", p1.apellidos, p1.nombre);

   return 0;
}
```

Si los campos de cadena de caracteres los leemos con *fgets()*, quedarán los '\n' residuales. Así sucede en el Ejemplo 7.

Ejemplo 7 Caracteres de fin de linea residuales en lecturas con *fgets()*.

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

typedef struct {
   char nombre[20];
   char apellidos[40];
} Persona;

int main() {

   Persona p1;

   printf("Nombre: ");
   fgets(p1.nombre, 20, stdin);
   printf("Apellidos: ");
   fgets(p1.apellidos, 40, stdin);

   // OJO, hay \n residual
   printf("%s, %s\n", p1.apellidos, p1.nombre);

   return 0;
}
```

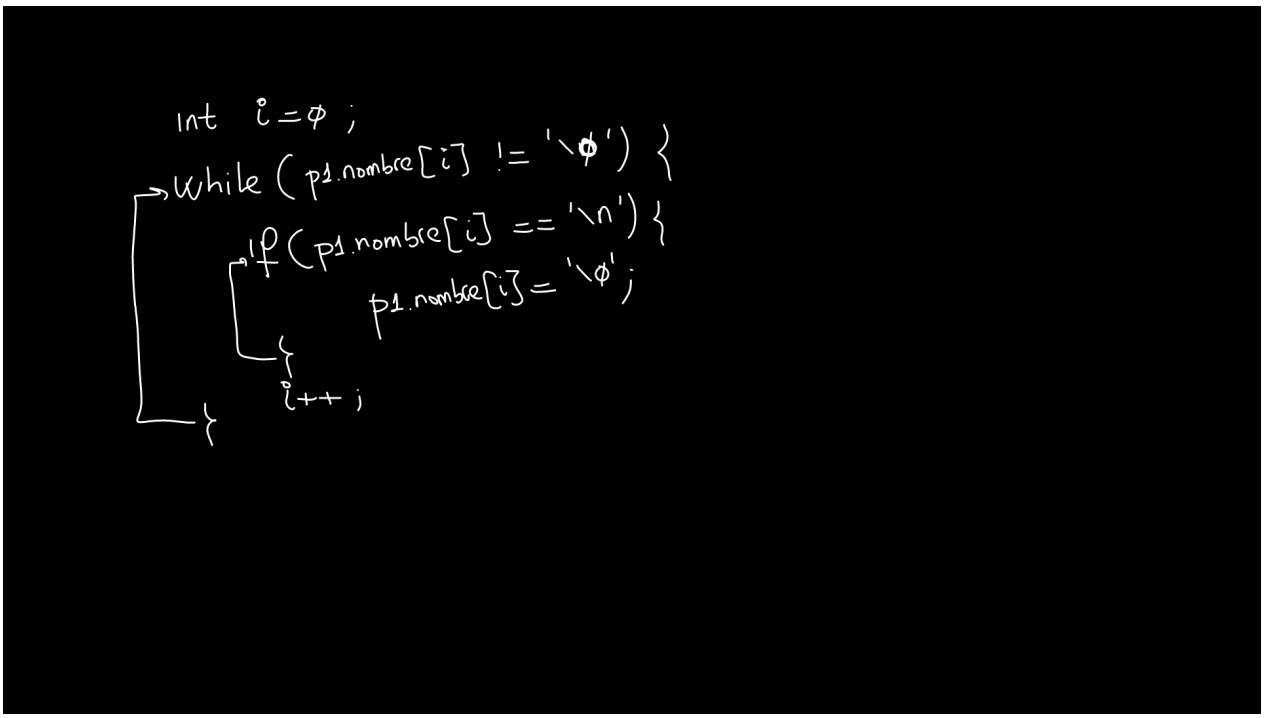

Figura 11: Eliminar el carácter '\n' residual en cadenas leídas desde la entrada estándar (método del bucle que recorre la cadena).

El Ejemplo 8 muestra varias funciones que pueden servir para eliminar los caracteres '\n' residuales de las cadenas.

Ejemplo 8 Funciones para *limpiar* los '\n' residuales.

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

// Definición de tipos
typedef struct {
   char nombre[20];
   char apellidos[40];
} Persona;

// Prototipos
void limpia_1(char* cad);
void limpia_2(char* cad);
void limpia_3(char* cad);

// Programa principal
int main() {
   Persona p1;

   printf("Nombre: ");
   fgets(p1.nombre, 20, stdin);
   limpia_1(p1.nombre);

   printf("Apellidos: ");
   fgets(p1.apellidos, 40, stdin);
   limpia_2(p1.apellidos);

   printf("%s, %s\n", p1.apellidos, p1.nombre);

   return 0;
}
```

```
// Código de las funciones
void limpia_1(char* cad) {
   int i=0;
   while(cad[i] != '\0') {
      if(cad[i] == '\n') {
         cad[i] = '\0';
      }
      i++;
   }
}
void limpia_2(char* cad) {
   int ultimo = strlen(cad)-1;
   if(cad[ultimo] == '\n') {
      cad[ultimo] = '\0';
   }
}
void limpia_3(char* cad) {
   for(int i=0; cad[i]!='\0'; i++) {
      if(cad[i] == '\n') {
         cad[i] = '\0';
      }
   }
}
```

El Ejemplo 9 muestra cómo copiar una cadena utilizando las funciones *strcpy()* y *strncpy()* de la biblioteca *string.h*.

Ejemplo 9 Ejemplo de uso de *strcpy()* y *strncpy()*

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

int main() {

   char cad1[] = "ABCDEF";

   char cad2[7];
   strcpy(cad2, cad1); // Copia la cadena completa, incluyendo el \0
   printf("%s %s\n", cad1, cad2);

   char cad3[7];
   strncpy(cad3, cad1, 3); // Copia solo 3 caracteres. No incluye el \0
   cad3[3] = '\0';
   printf("%s %s\n", cad1, cad3);

   return 0;
}
```

El Ejemplo 10 muestra la utilización de las funciones *strcmp()* y *strncmp()* de la biblioteca *string.h*, para comparar cadenas.

Ejemplo 10 Comparación de cadenas con *strcmp()* y *strncmp()*

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

int main() {

   char cad1[] = "ABCDEF";
   char cad2[] = "ABCDEF";
   printf("%d\n", strcmp(cad1, cad2));

   char cad3[] = "abcdef";
   printf("%d\n", strcmp(cad1, cad3));

   char cad4[] = "ABCXXX";
   printf("%d\n", strncmp(cad1, cad4, 3));

   return 0;
}
```

El Ejemplo 11 muestra la técnica para convertir caracteres de mayúsculas a minúsculas o viceversa.

Ejemplo 11 Conversiones mayúsculas-minúsculas

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

int main() {

   char cad1[] = "ABCDEF";

   cad1[0] = cad1[0] + 32;

   cad1[1] = cad1[1] + ('a' - 'A');

   printf("%s\n", cad1);

   return 0;
}
```

El Ejemplo 12 muestra una función que puede servir para convertir a mayúsculas los caracteres en minúsculas de una cadena.

Ejemplo 12 Función para convertir a mayúsculas

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

void a_mayuscula(char* cad);

int main() {
   char cad1[] = "abcZ";
   a_mayuscula(cad1);

   printf("%s\n", cad1);

   return 0;
}
void a_mayuscula(char* cad) {
   int i = 0;
   while (cad[i] != '\0') {
      if (cad[i] >= 'a' && cad[i] <= 'z') {
         cad[i] = cad[i] - 32;
      }
      i++;
   }
}
```

Sintaxis de puntero en la declaración de cadenas de caracteres

Si se declara e inicializa un array con sintaxis de puntero, el array es inmutable y no es posible modificarlo. El siguiente código da un error de violación de segmento al ejecutar la función *strcpy()*.

```
#include <stdio.h>

int main() {
   char* cad = "Prueba";
   printf("%s\n", cad);

   strcpy(cad, "PP"); // Violación de segmento
   printf("%s\n", cad);

   return 0;
}
```