

Informática II

Caracteres y cadenas

Gonzalo F. Perez Paina



Universidad Tecnológica Nacional
Facultad Regional Córdoba
UTN-FRC

– 2021 –

Cadenas

- ¿Existe el tipo de datos *string* (cadena) en C?

Cadenas

- ▶ ¿Existe el tipo de datos *string* (cadena) en C?
- ▶ ¿Cómo se almacenan/tratan las cadenas en C?

Cadenas

- ▶ ¿Existe el tipo de datos *string* (cadena) en C?
- ▶ ¿Cómo se almacenan/tratan las cadenas en C?

```
char cadena1[] = {67, 97, 100, 49, 0};  
printf("%s", cadera1); // ¿Qué se imprime?
```

Cadenas

- ▶ ¿Existe el tipo de datos *string* (cadena) en C?
- ▶ ¿Cómo se almacenan/tratan las cadenas en C?

```
char cadena1[] = {67, 97, 100, 49, 0};  
printf("%s", cadera1); // Imprime "Cad1"
```

Cadenas

- ▶ ¿Existe el tipo de datos *string* (cadena) en C?
- ▶ ¿Cómo se almacenan/tratan las cadenas en C?

En lenguaje C las *cadenas* son arreglos de caracteres (ASCII)

Cadenas

- ▶ ¿Existe el tipo de datos *string* (cadena) en C?
- ▶ ¿Cómo se almacenan/tratan las cadenas en C?

En lenguaje C las *cadenas* son arreglos de caracteres (ASCII)

Ejemplos:

```
char cadena1[] = {'C', 'a', 'd', 'a', '\0'};
```

Cadenas

- ▶ ¿Existe el tipo de datos *string* (cadena) en C?
- ▶ ¿Cómo se almacenan/tratan las cadenas en C?

En lenguaje C las *cadenas* son arreglos de caracteres (ASCII)

Ejemplos:

```
char cadena1[] = {'C', 'a', 'd', '1', '\0'};  
char cadena2[5] = {'C', 'a', 'd', '2', '\0'};
```


Cadenas

- ▶ ¿Existe el tipo de datos *string* (cadena) en C?
- ▶ ¿Cómo se almacenan/tratan las cadenas en C?

En lenguaje C las *cadenas* son arreglos de caracteres (ASCII)

Ejemplos:

```
char cadena1[] = {'C', 'a', 'd', '1', '\0'};  
char cadena2[5] = {'C', 'a', 'd', '2', '\0'};  
char cadena3[4] = {'C', 'a', 'd', '3', '\0'};
```

Cadenas

- ▶ ¿Existe el tipo de datos *string* (cadena) en C?
- ▶ ¿Cómo se almacenan/tratan las cadenas en C?

En lenguaje C las *cadenas* son arreglos de caracteres (ASCII)

Ejemplos:

```
char cadena1[] = {'C', 'a', 'd', 'a', '\0'};  
char cadena2[5] = {'C', 'a', 'd', 'a', '\0'};  
char cadena3[4] = {'C', 'a', 'd', 'a', '\0'}; // ERROR
```

Cadenas

- ▶ ¿Existe el tipo de datos *string* (cadena) en C?
- ▶ ¿Cómo se almacenan/tratan las cadenas en C?

En lenguaje C las *cadenas* son arreglos de caracteres (ASCII)

Ejemplos:

```
char cadena1[] = {'C', 'a', 'd', 'a', '\0'};  
char cadena2[5] = {'C', 'a', 'd', 'a', '\0'};  
char cadena3[4] = {'C', 'a', 'd', 'a', '\0'}; // ERROR  
char cadena4[] = {67, 97, 100, 52, 0}; // ASCII
```

Cadenas

- ▶ ¿Existe el tipo de datos *string* (cadena) en C?
- ▶ ¿Cómo se almacenan/tratan las cadenas en C?

En lenguaje C las *cadenas* son arreglos de caracteres (ASCII)

Ejemplos:

```
char cadena1[] = {'C', 'a', 'd', '1', '\0'};  
char cadena2[5] = {'C', 'a', 'd', '2', '\0'};  
char cadena3[4] = {'C', 'a', 'd', '3', '\0'}; // ERROR  
char cadena4[] = {67, 97, 100, 52, 0}; // ASCII  
char cadena5[] = "Cad5";
```

Cadenas

- ▶ ¿Existe el tipo de datos *string* (cadena) en C?
- ▶ ¿Cómo se almacenan/tratan las cadenas en C?

En lenguaje C las *cadenas* son arreglos de caracteres (ASCII)

Ejemplos:

```
char cadena1[] = {'C', 'a', 'd', 'a', '\0'};  
char cadena2[5] = {'C', 'a', 'd', 'a', '\0'};  
char cadena3[4] = {'C', 'a', 'd', 'a', '\0'}; // ERROR  
char cadena4[] = {67, 97, 100, 52, 0}; // ASCII  
char cadena5[] = "Cad5";  
char *cadena6 = "Cad6";
```

Cadenas

- ▶ ¿Existe el tipo de datos *string* (cadena) en C?
- ▶ ¿Cómo se almacenan/tratan las cadenas en C?

En lenguaje C las *cadenas* son arreglos de caracteres (ASCII)

Ejemplos:

```
char cadena1[] = {'C', 'a', 'd', '1', '\0'};  
char cadena2[5] = {'C', 'a', 'd', '2', '\0'};  
char cadena3[4] = {'C', 'a', 'd', '3', '\0'}; // ERROR  
char cadena4[] = {67, 97, 100, 52, 0}; // ASCII  
char cadena5[] = "Cad5";  
char *cadena6 = "Cad6";
```

Imprimir

```
printf("La cadena es: %s\n", cadena6);
```

Cadenas

- ▶ ¿Existe el tipo de datos *string* (cadena) en C?
- ▶ ¿Cómo se almacenan/tratan las cadenas en C?

En lenguaje C las *cadenas* son arreglos de caracteres (ASCII)

Ejemplos:

```
char cadena1[] = {'C', 'a', 'd', 'a', '\0'};  
char cadena2[5] = {'C', 'a', 'd', 'a', '\0'};  
char cadena3[4] = {'C', 'a', 'd', 'a', '\0'}; // ERROR  
char cadena4[] = {67, 97, 100, 52, 0}; // ASCII  
char cadena5[] = "Cad5";  
char *cadena6 = "Cad6";
```

Imprimir

```
printf("La cadena es: %s\n", cadena6);
```

- ▶ ASCII: American Standard Code for Information Interchange ([Tabla](#))
- ▶ Constante de caracter: 'a', '\0', '\n', etc.

Arreglo de caracteres

Archivo cadenas.c

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     char cadena1[] = {'C', 'a', 'd', '1', '\0'};
6     char cadena2[5] = {'C', 'a', 'd', '2', '\0'};
7     /* char cadena3[4] = {'C', 'a', 'd', '3', '\0'}; */
8     char cadena4[] = {67, 97, 100, 52, 0};
9     char cadena5[] = "Cad5";
10    char *cadena6 = "Cad6";
11
12    printf("Cadena 1: %s\n", cadena1);
13    printf("Cadena 2: %s\n", cadena2);
14    /* printf("Cadena 3: %s\n", cadena3); */
15    printf("Cadena 4: %s\n", cadena4);
16    printf("Cadena 5: %s\n", cadena5);
17    printf("Cadena 6: %s\n", cadena6);
18
19    return 0;
20 }
```

Cadenas en lenguaje C

Serie de caracteres tratados como una única unidad

Cadenas en lenguaje C

Serie de caracteres tratados como una única unidad

- En C, una cadena es un arreglo de caracteres terminado con el caracter nulo `'\0'`

Cadenas en lenguaje C

Serie de caracteres tratados como una única unidad

- ▶ En C, una cadena es un arreglo de caracteres terminado con el caracter nulo `'\0'`
- ▶ Puede incluir letras, dígitos y caracteres especiales como `+`, `-`, `*`, `/`, `$`, etc.

Cadenas en lenguaje C

Serie de caracteres tratados como una única unidad

- ▶ En C, una cadena es un arreglo de caracteres terminado con el caracter nulo `'\0'`
- ▶ Puede incluir letras, dígitos y caracteres especiales como `+`, `-`, `*`, `/`, `$`, etc.
- ▶ En C, las *literales* o *constantes* de cadenas se escriben en comillas dobles.

Cadenas en lenguaje C

Serie de caracteres tratados como una única unidad

- ▶ En C, una cadena es un arreglo de caracteres terminado con el caracter nulo `'\0'`
- ▶ Puede incluir letras, dígitos y caracteres especiales como `+`, `-`, `*`, `/`, `$`, etc.
- ▶ En C, las *literales* o *constantes* de cadenas se escriben en comillas dobles.
- ▶ Se tiene acceso mediante el puntero al primer caracter.

Cadenas en lenguaje C

Serie de caracteres tratados como una única unidad

- ▶ En C, una cadena es un arreglo de caracteres terminado con el caracter nulo `'\0'`
- ▶ Puede incluir letras, dígitos y caracteres especiales como `+`, `-`, `*`, `/`, `$`, etc.
- ▶ En C, las *literales* o *constantes* de cadenas se escriben en comillas dobles.
- ▶ Se tiene acceso mediante el puntero al primer caracter.

```
char color[] = "verde";  
char *ptrColor = "verde";
```

- ▶ Un arreglo de caracteres debe tener el tamaño adecuado para contener la cadena más el caracter de terminación.

Cadenas en lenguaje C

Serie de caracteres tratados como una única unidad

- ▶ En C, una cadena es un arreglo de caracteres terminado con el caracter nulo `'\0'`
- ▶ Puede incluir letras, dígitos y caracteres especiales como `+`, `-`, `*`, `/`, `$`, etc.
- ▶ En C, las *literales* o *constantes* de cadenas se escriben en comillas dobles.
- ▶ Se tiene acceso mediante el puntero al primer caracter.

```
char color[] = "verde";  
char *ptrColor = "verde";
```

- ▶ Un arreglo de caracteres debe tener el tamaño adecuado para contener la cadena más el caracter de terminación.

```
char cadena[20];  
scanf("%s", cadena);
```

Biblioteca para el manejo de caracteres

- ▶ Archivo de cabecera `<ctype.h>`
- ▶ Incluye funciones para realizar pruebas o verificación y para la manipulación de datos tipo caracteres.

Biblioteca para el manejo de caracteres

- ▶ Archivo de cabecera `<ctype.h>`
- ▶ Incluye funciones para realizar pruebas o verificación y para la manipulación de datos tipo caracteres.

Algunas funciones son:

- ▶ `int isdigit(int c)`

Biblioteca para el manejo de caracteres

- ▶ Archivo de cabecera `<ctype.h>`
- ▶ Incluye funciones para realizar pruebas o verificación y para la manipulación de datos tipo caracteres.

Algunas funciones son:

- ▶ `int isdigit(int c)`
- ▶ `int isalpha(int c)`

Biblioteca para el manejo de caracteres

- ▶ Archivo de cabecera `<ctype.h>`
- ▶ Incluye funciones para realizar pruebas o verificación y para la manipulación de datos tipo caracteres.

Algunas funciones son:

- ▶ `int isdigit(int c)`
- ▶ `int isalpha(int c)`
- ▶ `int isalnum(int c)`

Biblioteca para el manejo de caracteres

- ▶ Archivo de cabecera `<ctype.h>`
- ▶ Incluye funciones para realizar pruebas o verificación y para la manipulación de datos tipo caracteres.

Algunas funciones son:

- ▶ `int isdigit(int c)`
- ▶ `int isalpha(int c)`
- ▶ `int isalnum(int c)`
- ▶ `int isxdigit(int c)`

Biblioteca para el manejo de caracteres

- ▶ Archivo de cabecera `<ctype.h>`
- ▶ Incluye funciones para realizar pruebas o verificación y para la manipulación de datos tipo caracteres.

Algunas funciones son:

- ▶ `int isdigit(int c)`
- ▶ `int isalpha(int c)`
- ▶ `int isalnum(int c)`
- ▶ `int isxdigit(int c)`
- ▶ `int islower(int c)`

Biblioteca para el manejo de caracteres

- ▶ Archivo de cabecera `<ctype.h>`
- ▶ Incluye funciones para realizar pruebas o verificación y para la manipulación de datos tipo caracteres.

Algunas funciones son:

- ▶ `int isdigit(int c)`
- ▶ `int isalpha(int c)`
- ▶ `int isalnum(int c)`
- ▶ `int isxdigit(int c)`
- ▶ `int islower(int c)`
- ▶ `int isupper(int c)`

Biblioteca para el manejo de caracteres

- ▶ Archivo de cabecera `<ctype.h>`
- ▶ Incluye funciones para realizar pruebas o verificación y para la manipulación de datos tipo caracteres.

Algunas funciones son:

- ▶ `int isdigit(int c)`
- ▶ `int isalpha(int c)`
- ▶ `int isalnum(int c)`
- ▶ `int isxdigit(int c)`
- ▶ `int islower(int c)`
- ▶ `int isupper(int c)`
- ▶ `int tolower(int c)`

Biblioteca para el manejo de caracteres

- ▶ Archivo de cabecera `<ctype.h>`
- ▶ Incluye funciones para realizar pruebas o verificación y para la manipulación de datos tipo caracteres.

Algunas funciones son:

- ▶ `int isdigit(int c)`
- ▶ `int isalpha(int c)`
- ▶ `int isalnum(int c)`
- ▶ `int isxdigit(int c)`
- ▶ `int islower(int c)`
- ▶ `int isupper(int c)`
- ▶ `int tolower(int c)`
- ▶ `int toupper(int c)`

Funciones de conversión de cadenas

- ▶ Archivo de cabecera `<stdlib.h>` (biblioteca general de utilería).
- ▶ Convierte cadenas de dígitos a enteros y valores en punto flotante.

Funciones de conversión de cadenas

- ▶ Archivo de cabecera `<stdlib.h>` (biblioteca general de utilería).
- ▶ Convierte cadenas de dígitos a enteros y valores en punto flotante.

Algunas funciones son:

- ▶ `double` `atof(const char *nptr)`

Funciones de conversión de cadenas

- ▶ Archivo de cabecera `<stdlib.h>` (biblioteca general de utilería).
- ▶ Convierte cadenas de dígitos a enteros y valores en punto flotante.

Algunas funciones son:

- ▶ `double` `atof(const char *nptr)`
- ▶ `int` `atoi(const char *nptr)`

Funciones de conversión de cadenas

- ▶ Archivo de cabecera `<stdlib.h>` (biblioteca general de utilería).
- ▶ Convierte cadenas de dígitos a enteros y valores en punto flotante.

Algunas funciones son:

- ▶ `double` `atof(const char *nptr)`
- ▶ `int` `atoi(const char *nptr)`
- ▶ `long` `atol(const char *nptr)`

Funciones de conversión de cadenas

- ▶ Archivo de cabecera `<stdlib.h>` (biblioteca general de utilería).
- ▶ Convierte cadenas de dígitos a enteros y valores en punto flotante.

Algunas funciones son:

- ▶ `double` `atof(const char *nptr)`
- ▶ `int` `atoi(const char *nptr)`
- ▶ `long` `atol(const char *nptr)`
- ▶ `float` `strtof(const char *nptr, char **endptr)`

Funciones de conversión de cadenas

- ▶ Archivo de cabecera `<stdlib.h>` (biblioteca general de utilería).
- ▶ Convierte cadenas de dígitos a enteros y valores en punto flotante.

Algunas funciones son:

- ▶ `double` `atof(const char *nptr)`
- ▶ `int` `atoi(const char *nptr)`
- ▶ `long` `atol(const char *nptr)`
- ▶ `float` `strtof(const char *nptr, char **endptr)`
- ▶ `double` `strtod(const char *nptr, char **endptr)`

Funciones de conversión de cadenas

- ▶ Archivo de cabecera `<stdlib.h>` (biblioteca general de utilería).
- ▶ Convierte cadenas de dígitos a enteros y valores en punto flotante.

Algunas funciones son:

- ▶ `double` `atof(const char *nptr)`
- ▶ `int` `atoi(const char *nptr)`
- ▶ `long` `atol(const char *nptr)`
- ▶ `float` `strtof(const char *nptr, char **endptr)`
- ▶ `double` `strtod(const char *nptr, char **endptr)`

¿Qué significa `const char *nptr`?

Manipulación y comparación de cadenas

- ▶ Archivo de cabecera `<string.h>`

Manipulación y comparación de cadenas

- ▶ Archivo de cabecera `<string.h>`

Algunas funciones son:

- ▶ `char *strcpy(char *dest, const char *src)`

Manipulación y comparación de cadenas

- ▶ Archivo de cabecera `<string.h>`

Algunas funciones son:

- ▶ `char *strcpy(char *dest, const char *src)`
- ▶ `char *strncpy(char *dest, const char *src, size_t n)`

Manipulación y comparación de cadenas

- ▶ Archivo de cabecera `<string.h>`

Algunas funciones son:

- ▶ `char *strcpy(char *dest, const char *src)`
- ▶ `char *strncpy(char *dest, const char *src, size_t n)`
- ▶ `char *strcat(char *dest, const char *src)`

Manipulación y comparación de cadenas

- ▶ Archivo de cabecera `<string.h>`

Algunas funciones son:

- ▶ `char *strcpy(char *dest, const char *src)`
- ▶ `char *strncpy(char *dest, const char *src, size_t n)`
- ▶ `char *strcat(char *dest, const char *src)`
- ▶ `char *strncat(char *dest, const char *src, size_t n)`

Manipulación y comparación de cadenas

- ▶ Archivo de cabecera `<string.h>`

Algunas funciones son:

- ▶ `char *strcpy(char *dest, const char *src)`
- ▶ `char *strncpy(char *dest, const char *src, size_t n)`
- ▶ `char *strcat(char *dest, const char *src)`
- ▶ `char *strncat(char *dest, const char *src, size_t n)`
- ▶ `char *strcmp(const char *s1, const char *s2)`

Manipulación y comparación de cadenas

- ▶ Archivo de cabecera `<string.h>`

Algunas funciones son:

- ▶ `char *strcpy(char *dest, const char *src)`
- ▶ `char *strncpy(char *dest, const char *src, size_t n)`
- ▶ `char *strcat(char *dest, const char *src)`
- ▶ `char *strncat(char *dest, const char *src, size_t n)`
- ▶ `char *strcmp(const char *s1, const char *s2)`
- ▶ `char *strncmp(const char *s1, const char *s2, size_t n)`

Manipulación y comparación de cadenas

- ▶ Archivo de cabecera `<string.h>`

Algunas funciones son:

- ▶ `char *strcpy(char *dest, const char *src)`
- ▶ `char *strncpy(char *dest, const char *src, size_t n)`
- ▶ `char *strcat(char *dest, const char *src)`
- ▶ `char *strncat(char *dest, const char *src, size_t n)`
- ▶ `char *strcmp(const char *s1, const char *s2)`
- ▶ `char *strncmp(const char *s1, const char *s2, size_t n)`
- ▶ `size_t strlen(const char *s)`

Arreglos de punteros – Arreglos de cadenas

- ▶ Los arreglos pueden contener punteros

Arreglos de punteros – Arreglos de cadenas

- ▶ Los arreglos pueden contener punteros
- ▶ Uso común: arreglos de cadenas

Arreglos de punteros – Arreglos de cadenas

- ▶ Los arreglos pueden contener punteros
- ▶ Uso común: arreglos de cadenas
 - ▶ Cada entrada del arreglo es una cadena

Arreglos de punteros – Arreglos de cadenas

- ▶ Los arreglos pueden contener punteros
- ▶ Uso común: arreglos de cadenas
 - ▶ Cada entrada del arreglo es una cadena
 - ▶ Cada entrada es un puntero al primer caracter de la cadena

Arreglos de punteros – Arreglos de cadenas

- ▶ Los arreglos pueden contener punteros
- ▶ Uso común: arreglos de cadenas
 - ▶ Cada entrada del arreglo es una cadena
 - ▶ Cada entrada es un puntero al primer caracter de la cadena

Ejemplo

```
char *key[4] = {"Top", "Down", "Left", "Right"};
```

Arreglos de punteros – Arreglos de cadenas

- ▶ Los arreglos pueden contener punteros
- ▶ Uso común: arreglos de cadenas
 - ▶ Cada entrada del arreglo es una cadena
 - ▶ Cada entrada es un puntero al primer caracter de la cadena

Ejemplo

```
char *key[4] = {"Top", "Down", "Left", "Right"};
```

¿Qué pasa si se almacenan las cadenas en un arreglo bidimensional?

Ejemplo – Argumentos a la función `main`

```
int main(void)
{
    /* Programa */
    . . .
    return 0;
}
```

Ejemplo – Argumentos a la función `main`

```
int main(void)
{
    /* Programa */
    . . .
    return 0;
}
```

```
int main(int argc, char *argv[])
{
    /* Programa */
    . . .
    return 0;
}
```

Ejemplo – Argumentos a la función `main`

```
int main(void)
{
    /* Programa */
    . . .
    return 0;
}
```

```
int main(int argc, char *argv[])
{
    /* Programa */
    . . .
    return 0;
}
```

- ▶ `argc`: cantidad de argumentos en la línea de comandos al ejecutar el programa
- ▶ `argv`: puntero a un vector de cadenas que contiene los argumentos (`argv[argc]` es un puntero `NULL`)

