



@prog2ua

Tema 2: La clase `string`

Programación 2

Grado en Ingeniería Informática
Universidad de Alicante
Curso 2024-2025



1. Cadenas de caracteres en C
2. La clase `string` en C++
3. Conversiones de tipos
4. Comparativa
5. Ejercicios

Cadenas de caracteres en C

Declaración (1/3)

- Las *cadenas de caracteres* son vectores que contienen una secuencia de tipo `char` terminada en el carácter nulo (`'\0'`):

```
// El compilador mete el '\0' al final automáticamente  
char cad[]="hola";  
// Otra forma de inicializar, carácter a carácter  
char cad[]={ 'h', 'o', 'l', 'a', '\0' };  
// Falta el '\0': no es una cadena de caracteres válida  
char cad[]={ 'h', 'o', 'l', 'a' };
```

- Muchas de las funciones* que trabajan con cadenas buscan el `'\0'` para saber dónde termina la cadena
- Si no tenemos el `'\0'` puede que el resultado de estas funciones no sea el esperado

*Como aquellas que pertenecen a la librería `cstring` y que veremos más adelante

Declaración (2/3)

- Las cadenas de caracteres en C tienen tamaño fijo y una vez declaradas no pueden cambiar de tamaño:

```
char cad[10]; // Almacena como máximo 10 elementos
```

- Hay que tener en cuenta que se debe reservar siempre un espacio para almacenar el carácter nulo ('\\0'):

```
char cad[10]; // Almacena como máximo 9 letras y el '\\0'
```

- Se pueden inicializar al declararlas, en cuyo caso no hace falta poner el tamaño:

```
char cad[]="hola"; // Tamaño 5 (4 letras + '\\0')  
char cad2[10]="hola"; // Tamaño 10, aunque solo ocupa 5
```

- Las cadenas de caracteres en C se pueden usar también en C++

Declaración (3/3)

- Errores comunes al declarar cadenas de caracteres:

```
// El vector es demasiado pequeño para guardar la cadena  
char cad[5]="paralelepipedo"; // Error de compilación  
  
// Se usan comillas simples (') en lugar de dobles (")  
char cad[]='h'; // Error de compilación  
char cad[]='hola'; // Error de compilación  
  
// No se pone el tamaño y no se inicializa  
char cad[]; // Error de compilación  
  
// Se intenta asignar valor con '=' después de declarar  
char cad[10];  
cad="hola"; // Error de compilación
```

- Salida por pantalla con `cout` y `cerr` como el resto de tipos simples (`int`, `float`, etc.)
- Podemos combinar en la salida variables, constantes y datos de distinto tipo:

```
char cad[]="Nota";  
int num=10;  
  
cout << cad << " -> " << num; // Muestra "Nota -> 10"
```

Entrada por teclado > Operador >> (1/2)

- Podemos leer una cadena de caracteres desde teclado como con otros tipos simples, utilizando `cin` y el operador `>>`
- Existen algunas diferencias a la hora de leer desde teclado con respecto a otros tipos de datos
- Ignora los blancos* antes de la cadena:

```
char cad[32];  
cin >> cad;  
// El usuario escribe "  hola"  
// La variable cad almacena "hola"
```

*Entendemos por "blanco" un espacio, tabulador o salto de línea (' \n ')

Entrada por teclado > Operador >> (2/2)

- Termina de leer en cuanto encuentra el primer blanco en la cadena. **No nos permite leer entera una cadena que contenga blancos:**

```
char cad[32];  
cin >> cad;  
// El usuario escribe "buenas tardes"  
// La variable cad almacena "buenas"
```

- No limita el número de caracteres que se leen. **El usuario puede escribir una cadena más grande de lo que admite el vector:**

```
char cad[5];  
cin >> cad;  
// El usuario escribe "esternocleidomastoideo"  
// Puede invadir zonas de memoria que no debería y  
// producirse un fallo de segmentación
```

Entrada por teclado > getline (1/4)

- También podemos leer una cadena de caracteres de teclado mediante `cin` y la función `getline`
- Esta función permite leer cadenas con blancos y limitar el número de caracteres leídos:

```
const int TAM=100;
char cad[TAM];
// cad: variable donde almacenamos la cadena
// TAM: número de caracteres a leer
cin.getline(cad,TAM);
// Si el usuario introduce "buenas tardes"
// en cad se almacena "buenas tardes"
```

- Lee como máximo `TAM-1` caracteres o hasta que llegue al final de línea
- El `'\n'` del final de línea se lee pero no se guarda en la cadena
- La función añade `'\0'` al final de lo que ha leído (por eso sólo lee `TAM-1` caracteres)

Entrada por teclado > getline (2/4)

- Si el usuario introduce más caracteres de los que caben, estos se quedan en el *buffer* de teclado y la siguiente lectura falla:

```
char cad[10];  
cout << "Cadena 1: ";  
cin.getline(cad,10);  
cout << "Leído 1: " << cad << endl;  
cout << "Cadena 2: ";  
cin.getline(cad,10);  
cout << "Leído 2: " << cad << endl;
```

Terminal

```
$ miPrograma  
Cadena 1: hola a todo el mundo  
Leído 1: hola a to  
Cadena 2: Leído 2:
```

Entrada por teclado > getline (3/4)

- Pueden haber problemas cuando leemos de `cin` combinando el operador `>>` y la función `getline`:

```
int num;  
char cad[100];  
  
cout << "Num: ";  
cin >> num;  
cout << "Escribe una cadena: " ;  
cin.getline(cad,100);  
cout << "Lo que he leído es: " << cad << endl;
```

Terminal

```
$ miPrograma  
Num: 10  
Escribe una cadena: Lo que he leído es:
```

Entrada por teclado > getline (4/4)

- ¿Por qué sucede esto?
 - Con el operador >> se lee 10, pero se deja de leer cuando se encuentra el primer carácter no numérico ('\\n' en este caso)
 - Lo primero que encuentra en el *buffer* la función `getline` es un '\\n', por lo que termina de leer y no guarda nada en `cad`
- Solución:

```
...  
cin >> num;  
cin.ignore(); // Añadimos esta línea  
              // Saca el '\\n' del buffer  
// Ya se puede usar getline sin problema  
...
```

La librería `cstring` (1/3)

- La librería `cstring` contiene una serie de funciones que facilitan el trabajo con cadenas de caracteres
- Para poder utilizarla hay que incluir la librería en el código:

```
#include <cstring>
```

- `strlen` devuelve la longitud (número de caracteres) de una cadena:

```
char cad[10]="adios";  
cout << strlen(cad); // Imprime 5
```

- `strcpy` copia una cadena en otra. Hay que llevar cuidado de no superar el tamaño del vector de destino:

```
char cad[5];  
strcpy(cad,"hola"); // Cabe: 4 + '\0' = 5 caracteres  
strcpy(cad,"adios") // No cabe!! Violación de segmento
```

La librería `cstring` (2/3)

- `strcmp` compara dos cadenas en orden lexicográfico*, devolviendo 1 si `cad1>cad2`, 0 si `cad1==cad2` y -1 si `cad1<cad2`:

```
char cad1[]="adios";  
char cad2[]="adeu";  
cout << strcmp(cad1,cad2) << endl; // Imprime 1  
cout << strcmp(cad2,cad1) << endl; // Imprime -1  
cout << strcmp(cad1,cad1) << endl; // Imprime 0
```

- `strcat` añade el contenido de una cadena al final de otra. **Debe haber suficiente espacio en la cadena destino:**

```
char cad[10]="hola";  
strcat(cad,"", mu); // En total 9 caracteres (cabe)  
strcat(cad,"ndo"); // Añade 3 más (¡ya no cabe!)
```

*Orden que siguen las palabras en un diccionario

La librería `cstring` (3/3)

- Las funciones `strncpy`, `strncat` y `strncpy` comparan, copian o concatenan sólo los `n` primeros caracteres:

```
char cad[8];  
strncpy(cad, "hola, mundo", 4); // Solo copia "hola"  
cad[4]='\0'; // No añade el '\0' de manera automática  
// Lo hemos de añadir nosotros al final
```

```
char cad1[8]="adios";  
char cad2[8]="adeu";  
// Solo compara los dos primeros caracteres  
cout << strncmp(cad1, cad2, 2) << endl; // Imprime 0
```

```
char cad1[50]="Hola, ";  
char cad2[]="mundo maravilloso";  
strncat(cad1, cad2, 5); // cad1 valdrá "Hola, mundo"
```


Conversión a int y float

- Para pasar una cadena de caracteres a `int` o `float` se pueden usar las funciones `atoi` o `atof`
- Estas funciones pertenecen a la librería `cstdlib`:

```
#include <cstdlib> // Siempre que se vayan a usar

char cad[]="100";
int num=atoi(cad); // num vale 100

char cad2[]="10.5";
float num2=atof(cad2); // num2 vale 10.5
```

La clase `string` en C++

Definición (1/2)

- En C++ se pueden usar las cadenas de caracteres en C, pero además cuenta con la clase* `string` que permite trabajar de manera más cómoda y flexible con cadenas de caracteres:

```
// Declaración de una variable de tipo string  
string s; // No hay que indicar el tamaño de la cadena  
// Declaración con inicialización  
string s2="Alicante";  
// Declaración de una constante  
const string SALUDO="hola";
```

*Más información sobre lo que es una "clase" en el Tema 5

Definición (2/2)

- Un `string` tiene tamaño variable y puede crecer en función de las necesidades de almacenamiento del programa:

```
string s="hola"; // Almacena 4 caracteres  
s="hola a todo el mundo"; // Almacena 20 caracteres*  
s="ok"; // Almacena 2 caracteres
```

- No es necesario preocuparse del `'\0'`
- El paso de parámetros (valor y referencia) se hace como con cualquier tipo simple:

```
void miFuncion(string s1,string &s2){  
    // s1 se pasa por valor  
    // s2 se pasa por referencia  
}
```

*Un espacio en blanco cuenta como un carácter más

- Salida por pantalla con `cout` y `cerr` igual que con los vectores de caracteres en C:

```
string s="Nota";  
int num=10;  
  
cout << s << " -> " << num; // Muestra "Nota -> 10"
```

- Se puede leer de teclado con `cin` y el operador `>>` de la misma forma que con vectores de caracteres en C
- Ignora los blancos antes de la cadena y termina de leer cuando encuentra el primer blanco:

```
string s;  
cin >> s;  
// El usuario escribe "    hola"  
// La variable s almacena "hola"  
...  
// El usuario escribe "buenas tardes"  
// La variable s almacena "buenas"
```

Entrada por teclado > getline (1/2)

- Al igual que con los vectores de caracteres en C, podemos usar la función `getline` para leer cadenas
- Permite leer cadenas que contengan blancos:

```
string s;  
getline(cin,s);  
// Si el usuario introduce "buenas tardes"  
// en s se almacena "buenas tardes"
```

- No limita el número de caracteres que se leen, ya que con un `string` no es necesario
- ¡Ojo! Cambia la sintaxis con respecto a los vectores de caracteres en C

Entrada por teclado > `getline` (2/2)

- Si combinamos lecturas con el operador `>>` y `getline` tenemos el mismo problema que con los vectores de caracteres en C*
- Por defecto, `getline` lee hasta que encuentra el carácter salto de línea (`'\n'`)
- Podemos pasarle un parámetro adicional para indicar que lea hasta un determinado carácter:

```
string s;  
// Lee hasta que encuentra la primera coma  
getline(cin,s,',');  
// Lee hasta que encuentra el primer corchete  
getline(cin,s,'[');
```

*La solución es la misma que en la transparencia 11

Extraer palabras de un string

- Se pueden extraer palabras fácilmente de un `string` usando la clase `stringstream`:

```
#include <sstream> // Necesario si se usa stringstream
...
stringstream ss("Hola mundo cruel 666");
string s;

// En cada iteración del bucle lee hasta encontrar blanco
while(ss>>s){ // Extraemos las palabras una a una
    cout << "Palabra: " << s << endl;
}
```

Métodos de string (1/3)

- Al ser una clase, los métodos se invocan poniendo un punto tras el nombre de la variable
- `length` devuelve el número de caracteres de la cadena:

```
// unsigned int length()  
string s="hola, mundo";  
cout << s.length(); // Imprime 11
```

- `find` devuelve la posición en la aparece una subcadena dentro de una cadena:

```
// size_t find(const string &s,unsigned int pos=0)  
cout << s.find("mundo"); // Imprime 6  
// Si no encuentra la subcadena devuelve string::npos
```

Métodos de string (2/3)

- `replace` sustituye una cadena (o parte de ella) por otra:

```
// string& replace(unsigned int pos,unsigned int tam,  
    const string &s)  
string s="hola mundo";  
s.replace(0,4,"hello"); // s vale "hello mundo"
```

- `erase` permite eliminar parte de una cadena:

```
// string& erase(unsigned int pos=0,unsigned int tam=  
    string::npos);  
string cad="hola mundo";  
cad.erase(4,3); // cad vale "holando"
```

- `substr` devuelve una subcadena de la cadena original:

```
// string substr(unsigned int pos=0,unsigned int tam=  
    string::npos) const;  
string cad="hola mundo";  
string subcad=cad.substr(2,5); // subcad vale "la mu"
```

Métodos de string (3/3)

- Ejemplo de uso:

```
string a="Hay una taza en esta cocina con tazas";
string b="taza";
unsigned int tam=a.length(); // Longitud de a
// Buscamos la primera palabra "taza"
size_t encontrado=a.find(b);
if(encontrado!=string::npos){
    cout << "Primera en: " << encontrado << endl;
    // Buscamos la segunda palabra "taza"
    encontrado=a.find(b,encontrado+b.length());
    if(encontrado!=string::npos)
        cout << "Segunda en: " << encontrado << endl;
}
else{
    cout << "Palabra '" << b << "' no encontrada";
}
// Sustituimos la primera "taza" por "botella"
a.replace(a.find(b),b.length(),"botella");
cout << a << endl;
```

Operadores (1/2)

- Comparaciones: == (igual), != (distinto), > (mayor estricto), >= (mayor o igual), < (menor estricto) y <= (menor o igual)

```
string s1,s2;  
cin >> s1; cin >> s2;  
if(s1==s2) // La comparación es en orden lexicográfico  
    cout << "Son iguales" << endl;
```

- Asignación de una cadena a otra con el operador =, como cualquier tipo simple:

```
string s1="hola";  
string s2;  
s2=s1;
```

- Concatenación de cadenas con el operador +:

```
string s1="hola";  
string s2="mundo";  
string s3=s1+", "+s2; // s3 vale "hola, mundo"
```

Operadores (2/2)

- Acceso a componentes como si fuera un vector de caracteres en C, con el operador []:

```
string s="hola";  
char c=s[3]; // s[3] vale 'a'  
s[0] = 'H';  
cout << s << ":" << c << endl ; // Imprime "Hola:a"
```

- No se pueden asignar caracteres a posiciones que no pertenecen al string:

```
string s;  
s[0]='h'; s[1]='o'; s[2]='l'; s[3]='a';  
// No almacena nada, porque s es una cadena vacía y esas  
posiciones no las tiene reservadas
```

- Ejemplo de recorrido de un string carácter a carácter:

```
string s="hola, mundo";  
for(unsigned int i=0;i<s.length();i++)  
    s[i]='f'; // Sustituye cada carácter por 'f'
```

Conversiones de tipos

Conversión entre `string` y vector de caracteres en C

- Para asignar un vector de caracteres en C a `string` se utiliza el operador de asignación (=):

```
char cad[]="hola";  
string s;  
s=cad;
```

- Para asignar un `string` a un vector de caracteres en C hay que usar `strcpy` y `c_str`.*

```
char cad[10];  
string s="mundo";  
// Debe haber espacio suficiente en cad  
strcpy(cad,s.c_str());
```

*El método `c_str` devuelve un vector de caracteres en C con el contenido del `string`

Conversión entre `string` y número

- Convertir un número entero o real a `string`:

```
#include <string> // ¡Ojo! No es lo mismo que <cstring>
...
int num=100;
string s=to_string(num);
```

- Convertir un `string` a número entero:*

```
string s="100";
int num=stoi(s);
```

- Convertir un `string` a número real:

```
string s="10.5";
float num=stof(s);
```

*`to_string`, `stoi` y `stof` están disponibles a partir de la versión 2011 de C++

Comparativa

Vector de caracteres en C vs. string

Vector de caracteres en C	string
<pre>char cad[TAM]; char cad[]="hola"; strlen(cad) cin.getline(cad,TAM); if(!strcmp(cad1,cad2)){...} strcpy(cad1,cad2); strcat(cad1,cad2); strcpy(cad,s.c_str());</pre>	<pre>string s; string s="hola"; s.length() getline(cin,s); if(s1==s2){...} s1=s2; s1=s1+s2; s=cad;</pre>
Terminan con ' <code>\0</code> '	No terminan con ' <code>\0</code> '
Tamaño reservado fijo	El tamaño reservado puede variar
Tamaño ocupado variable	Tamaño ocupado == tamaño reservado
Se usan con ficheros binarios	No se pueden usar con ficheros binarios

Ejercicios

Ejercicios (1/4)

Ejercicio 1

Diseña una función `subCadena` que devuelva la subcadena de longitud `n` que empieza en la posición `p` de otra cadena. Tanto el argumento como el valor de retorno deben ser de tipo `string`.

```
subCadena("hoooola",2,5) // Devuelve "la"
```

Ejercicio 2

Diseña una función `borraCaracterCadena` que, dados un `string` y un carácter, borre todas las apariciones del carácter en el `string` y lo devuelva.

```
borrarCaracterCadena("cocobongo",'o') // Devuelve "ccbng"
```

Ejercicio 3

Diseña una función `buscarSubcadena` que busque la primera aparición de una subcadena `a` dentro de una cadena `b` y devuelva su posición, o `-1` si no está. Tanto `a` como `b` deben ser de tipo `string`.

```
buscarSubcadena("ool", "hoooola") // Devuelve 2
```

Ampliaciones:

1. Añadir otro parámetro a la función que indique el número de aparición (si vale 1 sería como la función original)
2. Crear otra función que devuelva el número de apariciones de la subcadena en la cadena

Ejercicios (3/4)

Ejercicio 4

Diseña una función `codifica` que codifique una cadena sumando una cantidad `n` al código ASCII de cada carácter, pero teniendo en cuenta que el resultado debe ser una letra.

Por ejemplo, si `n=3`, la `a` se codifica como `d`, la `b` como `e`,..., la `x` como `a`, la `y` como `b`, y la `z` como `c`.

La función debe admitir letras mayúsculas y minúsculas. Los caracteres que no sean letras no se deben codificar. El argumento debe ser de tipo `string`.

```
codifica("hola, mundo",3) // Devuelve "krod, pxqgr"
```

Ejercicios (4/4)

Ejercicio 5

Diseña una función `esPalindromo` que devuelva `true` si el `string` que se le pasa como parámetro es palíndromo.

```
esPalindromo("larutanatural") // Devuelve true  
esPalindromo("hola, aloh") // Devuelve false
```

Ejercicio 6

Diseña una función `crearPalindromo` que añada a un `string` el mismo `string` invertido, de forma que el resultado sea un palíndromo.

```
crearPalindromo("hola") // Devuelve "holaaloh"
```