



@prog2ua

Tema 2: La clase `string`

Programación 2

Grado en Ingeniería Informática
Universidad de Alicante
Curso 2025-2026



1. La clase `string` en C++
2. Vectores STL
3. Argumentos del programa
4. Ejercicios

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 diapositiva 44 del Tema 1

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'
```

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++

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

Vectores STL

Vectores STL (1/3)

- La *Standard Template Library* (STL) es una librería de funciones para C++
- Proporciona diferentes estructuras de datos y algoritmos
- Incluye la clase `vector`, que permite almacenar elementos de cualquier tipo, como un vector normal, pero sin tener que preocuparnos del tamaño:

```
#include <vector> // Siempre que vayamos a usar vector
vector<int> vec; // Declara un vector de enteros
                // No es necesario indicar su tamaño
```

- El tamaño inicial de un vector STL es 0 y crece de manera dinámica en función de las necesidades
- Para añadir elementos al final del vector usamos `push_back`:*

```
vec.push_back(12); // Añade 12 al final del vector
vec.push_back(8); // Añade 8 detrás del 12
```

*Al ser una clase, sus métodos se invocan poniendo un punto tras el nombre de la variable

Vectores STL (2/3)

- Acceso a elementos mediante el operador []:

```
vec[10]=23; // Igual que un vector convencional  
cout << vec[8] << endl;
```

- Con `size` obtenemos el número de elementos del vector:

```
// Recorremos todos los elementos del vector  
for(unsigned int i=0;i<vec.size();i++){  
    vec[i]=10;  
}
```

- Recorrido de vectores basado en rango:

```
// Recorremos todos los elementos del vector  
for(int num:vec){  
    cout << num <<endl;  
}
```

Vectores STL (3/3)

- Con `clear` podemos borrar todos los elementos y con `erase` uno en concreto:

```
vec.erase(vec.begin()+3); // Elimina el cuarto elemento  
vec.clear(); // Elimina todos los elementos del vector
```

- Error habitual: **no se pueden guardar elementos en posiciones que no pertenecen al vector**

```
vector<int> vec;  
vec[0]=78; vec[1]=9; vec[2]=17;  
for(int i=0;i<vec.size();i++){  
    cout << vec[i] << endl; // No imprime nada  
}
```

- Existen muchas otras funciones para trabajar con vectores STL *

*Más información en <http://www.cplusplus.com/reference/vector/vector/>

Argumentos del programa

Argumentos del programa (1/4)

- Los *argumentos* de un programa se usan para proporcionarle información (normalmente opciones) desde línea de comandos
- Su uso es muy habitual y permite modificar el comportamiento del programa:

Terminal

```
$ ls          // Muestra los ficheros de un directorio
$ ls -a       // Muestra también los ficheros ocultos (opción "-a")
$ ls -a -l    // Añade información extra de cada fichero (opción "-l")
```


Argumentos del programa (2/4)

- El `main` es una función y como tal puede recibir dos parámetros: `argc` y `argv`
- Estos parámetros permiten gestionar el paso de argumentos por línea de comandos a nuestro programa:

```
// Siempre en este orden  
int main(int argc, char *argv[]) {  
    ...  
    return 0;  
}
```

- `int argc`: número de argumentos pasados al programa (contando también el nombre del programa)
- `char *argv[]`: vector de cadenas de caracteres con los argumentos pasados al programa

Argumentos del programa (3/4)

- Ejemplo de uso:

```
int main(int argc, char *argv[]){  
    for(int i=0; i<argc; i++){  
        cout << "Arg. " << i << " : " << argv[i] << endl;  
    }  
}
```

Terminal

```
$ ./miPrograma -a -h X    // Ejemplo de llamada con tres parámetros  
Arg. 0 : ./miPrograma  
Arg. 1 : -a  
Arg. 2 : -h  
Arg. 3 : X
```

- Los argumentos no tienen por qué empezar con un guión (-) pero es una práctica bastante habitual

Argumentos del programa (4/4)

- Parece fácil gestionar los argumentos del programa, pero a veces puede ser complicado
- El usuario no siempre usa el mismo orden a la hora de introducir los argumentos:

Terminal

```
$ g++ -Wall -o prog prog.cc -g  
$ g++ -g -Wall prog.cc -o prog
```

- Puede haber errores en la introducción y hay que mostrar mensajes de ayuda al usuario
- Es recomendable usar una función aparte para gestionar los argumentos

Ejercicios

Ejercicios (1/5)

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 (3/5)

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 (4/5)

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"
```


Ejercicios (5/5)

Ejercicio 7

Implementa un programa que contenga una función con el siguiente prototipo: `int primeNumber(int n)`. Esta función devolverá el *n*-ésimo número primo. El programa debe imprimir números primos por pantalla con las siguientes opciones:

- `-L` imprimir cada número en una línea distinta (por defecto se imprimen todos en la misma línea)
- `-N n` imprimir los *n* primeros números primos (por defecto 10)

Ejemplos de ejecución:

Terminal

```
$ primes -N 5
1 2 3 5 7
$ primes -N -L 5
Error: primes [-L] [-N n]
```