

Algoritmos y

Estructuras

De

Datos

ESTRUCTURAS

DE DATOS:

Strings

Cadenas de Caracteres en C++: el Tipo string

El lenguaje C++ dispone de la **biblioteca estándar <string>**, que proporciona el **tipo string** para representar **cadenas de caracteres**.

El tipo string dispone de **operadores predefinidos** que permiten manejar cadenas de caracteres de forma muy simple e intuitiva.

- ✓ Sin longitud máxima (gestión automática de la memoria)
- ✓ Multitud de funciones de utilidad (biblioteca string)

El tipo string puede ser utilizado para definir **constantes simbólicas, variables o parámetros formales** en los subprogramas.

- ❑ Aunque, **en AEDD no emplearemos datos de tipo string** como elementos (campos) de una estructura (struct) en Archivos.

Los strings de C++ **hacen crecer el espacio de almacenamiento** para acomodarse a los cambios de tamaño de los datos de la cadena por encima de los límites de la memoria asignada inicialmente.

Cadenas de Caracteres en C++: el Tipo string

■ Declaración e Inicialización de variables tipo string

```
#include <iostream>
#include <string>
using namespace std;
const string AUTOR = "Jose Luis";

int main() {
    string nombre = "Pepe";
    cout << "Nombre: " << nombre << endl;
    nombre = AUTOR;
    cout << "Nombre: " << nombre << endl;
    return 0;
}
```

AUTOR:	J	o	s	e		L	u	i	s
	0	1	2	3	4	5	6	7	8
nombre:	P	e	p	e					
	0	1	2	3					
nombre:	J	o	s	e		L	u	i	s
	0	1	2	3	4	5	6	7	8

```
Nombre: Pepe
Nombre: Jose Luis
```

Si la definición de una variable de tipo string no incluye la asignación de un valor inicial, dicha variable tendrá como valor **por defecto la cadena vacía ("")**.

Entrada y Salida de Cadenas tipo String

La entrada/salida de datos de tipo string se basa en el uso de los operadores **>>** y **<<** **sobre los flujos cin y cout.**

El operador **<<** sobre un flujo de salida cout muestra todos los caracteres que forman parte de la cadena.

El operador de entrada (**>>**) se comporta de la siguiente forma:

- ▶ **Elimina los espacios en blanco que hubiera al principio** de la entrada de datos

- ▶ Y lee dicha entrada **hasta que encuentre algún carácter de espacio en blanco**, que no será leído y permanecerá en el buffer de entrada hasta la próxima operación de entrada.

Como consecuencia de lo anterior, **no es posible utilizar el operador >> para leer una cadena de caracteres que incluya algún carácter en blanco.**

Entrada y Salida de Cadenas tipo String

No es posible utilizar el operador >> para leer una cadena de caracteres que incluya algún carácter en blanco.

El carácter en blanco actúa como delimitador y fuerza el fin de la lectura.

Ejemplo:

```
#include <iostream>
#include <string>
using namespace std;
```

```
int main() {
    string nombre;
    cout << "Introduzca el nombre: ";
    cin >> nombre;
    cout << "Nombre: " << nombre << endl;
    return 0;
}
```

```
Introduzca el nombre: Maria Luisa
Nombre: Maria
```

Tipo String: Entrada de Cadenas

Si se desea leer una secuencia de caracteres que incluya espacios en blanco, utilizaremos **la función getline en lugar del operador >>**.

- **getline()** lee y almacena en una **variable de tipo string** todos los caracteres del buffer de entrada, hasta leer el carácter de fin de línea (ENTER). No elimina espacios iniciales.

```
#include <iostream>
#include <string>
using namespace std;
```

```
int main() {
    string nombre;
    cout << "Introduzca el nombre: ";
    getline(cin, nombre);
    cout << "Nombre: " << nombre << endl;
    return 0;
}
```



```
Introduzca el nombre: Juan Antonio
Nombre: Juan Antonio
```

Tipo String: Entrada de Cadenas

Además, la función **getline** permite especificar el **delimitador que marca el final de la secuencia de caracteres a leer**.

Si no se especifica ninguno, por defecto se utiliza el carácter de fin de línea. Sin embargo, si se especifica el delimitador, **lee y almacena todos los caracteres del buffer hasta leer el carácter delimitador especificado, el cual es eliminado del buffer, pero no es almacenado en la variable de getline**.

En el siguiente ejemplo se utiliza un punto como delimitador en getline, por lo que la lectura de teclado acaba cuando se localice dicho carácter.

```
const char DELIMITADOR = '.';
```

```
int main() {  
    string nombre;  
    cout << "Introduzca el nombre: ";  
    getline(cin, nombre, DELIMITADOR);  
    cout << "Nombre: " << nombre << endl;  
    return 0;  
}
```

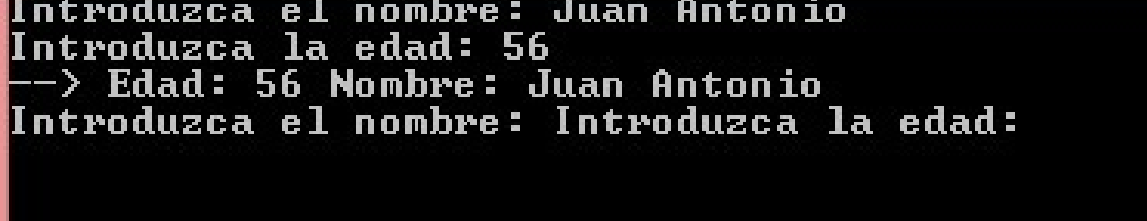
```
Introduzca el nombre: Juan Anto.nio  
Nombre: Juan Anto
```


Problemas con getline()

En ocasiones, cuando se utiliza una lectura con `getline`, después de una lectura previa con `>>`, podemos encontrarnos con un comportamiento que, aunque correcto, puede no corresponder al esperado intuitivamente.

```
#include <iostream>
#include <string>
using namespace std;

int main(){
    string nombre;
    int edad;
    for(int i = 0; i<5; i++){
        cout << "Introduzca el nombre: ";
        getline(cin, nombre);
        cout << "Introduzca la edad: ";
        cin >> edad;
        cout << "--> Edad: " << edad << " Nombre: " << nombre << endl;
    }
    return 0;
}
```



Introduzca el nombre: Juan Antonio
Introduzca la edad: 56
--> Edad: 56 Nombre: Juan Antonio
Introduzca el nombre: Introduzca la edad:

Problemas con getline()

```
int main(){
    string nombre;
    int edad;
    for(int i = 0; i<5; i++){
        cout << "Introduzca el nombre: ";
        getline(cin, nombre);
        cout << "Introduzca la edad: ";
        cin >> edad;
        cout << "--> Edad: " << edad << " Nombre: " << nombre << endl;
    }
    return 0;
}
```

La primera iteración funciona adecuadamente.

Las siguientes iteraciones funcionan de forma anómala, ya que la ejecución del programa no se detiene para que el usuario pueda introducir el nombre.

Hay que considerar que después de leer la edad en una determinada iteración, en el buffer permanece el carácter de fin de línea (ENTER) que se introdujo tras teclear la edad, ya que éste no es leído por el operador >> y permanece en el buffer de entrada.

En la siguiente iteración, la función getline –que lee una secuencia de caracteres hasta encontrar un ENTER (sin saltar los espacios iniciales)–, leerá el carácter ENTER que quedó en el buffer en la lectura previa de la edad de la iteración anterior, haciendo que finalice la lectura directamente.

Solución: limpiar el buffer - ws

El resultado es que, al leer el nombre, se lee una cadena vacía, sin que se detenga el programa para que el usuario introduzca el nombre de la persona.

La solución a este problema es eliminar los caracteres de espacios en blanco (y fin de línea) del buffer de entrada. De esta forma el buffer estará realmente vacío y conseguiremos que la ejecución de `getline()` haga que el programa se detenga hasta que el usuario introduzca el nombre.

Hay diferentes formas de conseguir que el buffer se quede vacío.

- Utilizaremos el **manipulador ws en el flujo cin**, que **extrae todos los espacios en blanco hasta encontrar algún carácter distinto** (en ese momento la extracción se detiene).

```
int main() {
    string nombre;
    int edad;
    for(int i = 0; i < 5; i++) {
        cout << "Introduzca el nombre: ";
        cin >> ws; //Elimina los espacios en blanco y el fin de línea
        getline(cin, nombre);
        cout << "Introduzca la edad: ";
        cin >> edad;
        cout << "--> Edad: " << edad << " Nombre: " << nombre << endl;
    }
    return 0;
}
```

```
Introduzca el nombre: Juan Antonio
Introduzca la edad: 56
--> Edad: 56 Nombre: Juan Antonio
Introduzca el nombre: Luciana
Introduzca la edad: 48
--> Edad: 48 Nombre: Luciana
Introduzca el nombre:
```

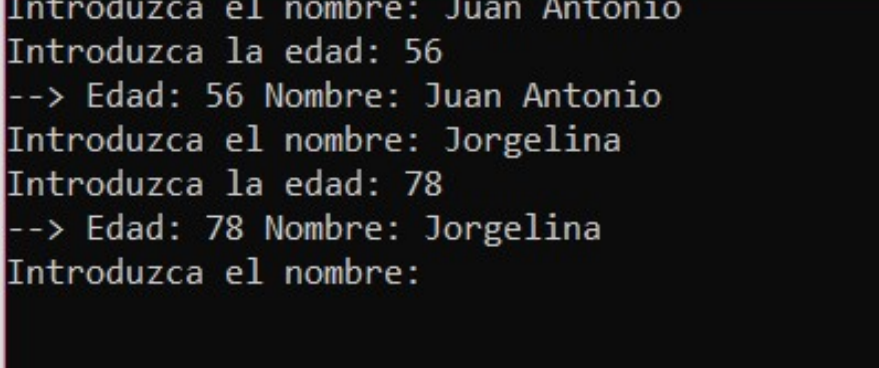
Solución: limpiar el buffer – *ignore()*

Para eliminar los caracteres que pudiera contener el buffer (no únicamente espacios en blanco) después de la última operación de lectura de datos, usamos la **función `cin.ignore()`**.

- **`cin.ignore()`** elimina todos los caracteres del buffer de entrada en el flujo especificado, hasta que se haya eliminado el número de caracteres indicado en el primer argumento o bien se haya eliminado el carácter indicado en el segundo.

La sentencia **`cin >> ws`** se asocia a la función `getline` que le sigue, mientras que la sentencia **`cin.ignore` se asocia a la sentencia de entrada `>>`** que aparece antes.

```
int main(){
    string nombre;
    int edad;
    for(int i = 0; i<5; i++){
        cout << "Introduzca el nombre: ";
        getline(cin, nombre);
        cout << "Introduzca la edad: ";
        cin >> edad;
        cin.ignore(10000, '\n'); //elimina todos los caracteres del buffer hasta '\n'
        cout << "--> Edad: " << edad << " Nombre: " << nombre << endl;
    }
    return 0;
}
```



```
Introduzca el nombre: Juan Antonio
Introduzca la edad: 56
--> Edad: 56 Nombre: Juan Antonio
Introduzca el nombre: Jorgelina
Introduzca la edad: 78
--> Edad: 78 Nombre: Jorgelina
Introduzca el nombre:
```

Comparación / Concatenación / Longitud

- Comparaciones lexicográficas con **operadores relacionales** (`==`, `!=`, `>`, `<`, `>=`, `<=`):

```
if (nombre >= AUTOR) { /*...*/ }
```

- **+**: Concatenación de cadenas:

```
const string AUTOR = "José Luis" ;  
int main ()  
{  
    string nombre = AUTOR + "López" ;  
    nombre += "Vázquez" ;  
    nombre += 'z' ;  
    nombre = AUTOR + 's' ;  
}
```

- **size()** o **length()**: obtención de la longitud de la cadena (número de caracteres):

```
unsigned ncar = nombre.size();  
if (nombre.size() == 0) { /*...*/ }
```

`nombre.length()`

Acceso a los caracteres de una cadena

- **Utilizando índices:**

```
char c = nombre[i]; donde  $i \in [0..\text{nombre.size()}-1]$   
nombre[i] = 'z'; donde  $i \in [0..\text{nombre.size()}-1]$ 
```

El índice debe corresponder a una posición válida de la cadena.

No tiene control de acceso a posiciones inexistentes del array (C++ no nos avisará cuando suceda).

- **Mediante la función `at` (índice):** Devuelve el carácter en la posición especificada y lanzará una excepción si se accede a una posición inexistente.

```
int main() {  
    string cadena1="Algoritmos";  
    cout << cadena1.at(2) << endl;  
    cout << cadena1.at(17) << endl;  
    return 0;  
}
```

Funciona igual que:
`cout << cadena1[2] << endl;`
`cout << cadena1[17] << endl;`
Aunque lanza una excepción si hay algo mal.

```
g  
terminate called after throwing an instance of 'std::out_of_range'  
  what():  basic_string::at: __n (which is 17) >= this->size() (which is 10)  
  
<< El programa ha finalizado: codigo de salida: 3 >>  
<< Presione enter para cerrar esta ventana >>
```

Operaciones con cadenas tipo string (1)

- **substr (posición, longitud):** obtiene una **nueva subcadena** a partir de *posición*

```
string cad = "abcdefg";  
cout << cad.substr(2, 3); // Muestra cde
```

Si **no se especifica** una *longitud*, o si esta **excede** al número de caracteres que hay desde *posición*, entonces se devuelve la **subcadena desde i hasta el final**.

- **swap (cadena2):** intercambia *cadena2* con otra cadena

```
int main() {  
    string cadena1="Hola";  
    string cadena2="Adios";  
    cout << "*** Antes del cambio ***" << endl;  
    cout << "Cadena1: " << cadena1 << endl;  
    cout << "Cadena2: " << cadena2 << endl << endl;  
    cadena1.swap(cadena2);  
    cout << "*** Despues del cambio ***" << endl;  
    cout << "Cadena1: " << cadena1 << endl;  
    cout << "Cadena2: " << cadena2 << endl << endl;  
    return 0;  
}
```

```
*** Antes del cambio ***  
Cadena1: Hola  
Cadena2: Adios  
  
*** Despues del cambio ***  
Cadena1: Adios  
Cadena2: Hola
```

Operaciones con cadenas tipo string (2)

- **find (subcadena):** devuelve la **posición** de la **1era ocurrencia** de *subcadena* en la cadena

```
string cad = "Olala";  
cout << cad.find("la"); // Muestra 1
```

- **rfind (subcadena):** devuelve la **posición** de la **última ocurrencia** de *subcadena* en la cadena

```
string cad = "Olala";  
cout << cad.rfind("la"); // Muestra 3
```


Eliminación / Inserción

- **erase (posición, cantidad):** elimina *cantidad* caracteres, a partir de *posición*

```
string cad = "abcdefgh";  
cad.erase(3, 4); // cad ahora contiene "abch"
```

Si no se especifica una *cantidad*, entonces se elimina **hasta el final** de la cadena.

- **insert (posición, cadena2):** inserta *cadena2*, a partir de *posición*

```
string cad = "abcdefgh";  
cad.insert(3, "123"); // cad ahora contiene "abc123defgh"
```

Pasaje de strings a funciones

```
#include <iostream>
using namespace std;

bool validar(string cadena);

int main(int argc, char *argv[]) {
    string cadena;
    int resultado;
    cout << "Ingrese una cadena: ";
    getline(cin, cadena);
    resultado = validar(cadena);
    if (resultado == 1)
        cout << "La cadena está vacía";
    else
        cout << "La cadena no está vacía";
    return 0;
}
```

Ingrese una cadena:
La cadena está vacía

Ingrese una cadena: UTN Santa Fe
La cadena no está vacía

```
bool validar(string cadena){
    if(cadena.size()==0)
        return true;
    else return false;
}
```

Pasaje por copia o valor
(opción por defecto)

Pasaje de strings a funciones

El pasaje de strings a funciones puede realizarse de 2 maneras: **por copia** y **por referencia**.

```
#include <iostream>
using namespace std;

void mayusculas(string& palabra);
```

```
int main(int argc, char *argv[]) {
    string palabra;
    cin >> palabra;
    cout << "Antes de pasar a mayúsculas: " << palabra << endl;
    mayusculas(palabra);
    cout << "Luego de pasar a mayúsculas: " << palabra << endl;
    return 0;
}
```

```
void mayusculas(string& palabra){
    for(unsigned i=0; i < palabra.size(); i++)
        palabra[i]=char(toupper(palabra[i]));
}
```

```
facultad
Antes de pasar a mayúsculas: facultad
Luego de pasar a mayúsculas: FACULTAD
```

**Pasaje por
referencia**

Una biblioteca para manejar caracteres

La biblioteca **<cctype>** proporciona principalmente operaciones sobre los valores de tipo char:

Prototipo de la función	Descripción	Ejemplo
<code>int isalpha(charExp)</code>	Devuelve verdadero (número entero diferente de cero) si <code>charExp</code> evalúa una letra; de lo contrario, devuelve falso (número entero cero)	<code>isalpha('a')</code>
<code>int isalnum(charExp)</code>	Devuelve verdadero (número entero diferente de cero) si <code>charExp</code> evalúa una letra o un dígito; de lo contrario, devuelve falso (número entero cero)	<code>char key;</code> <code>cin >> key;</code> <code>isalnum(key);</code>
<code>int isupper(charExp)</code>	Devuelve verdadero (número entero diferente de cero) si <code>charExp</code> evalúa una letra mayúscula; de lo contrario, devuelve falso (número entero cero)	<code>isupper('A')</code>
<code>int islower(charExp)</code>	Devuelve verdadero (número entero diferente de cero) si <code>charExp</code> evalúa una letra minúscula; de lo contrario, devuelve falso (número entero cero)	<code>islower('a')</code>
<code>int isdigit(charExp)</code>	Devuelve verdadero (número entero diferente de cero) si <code>charExp</code> evalúa un dígito (0 a 9); de lo contrario, devuelve falso (número entero cero)	<code>isdigit('5')</code>
<code>int isspace(charExp)</code>	Devuelve verdadero (número entero diferente de cero) si <code>charExp</code> evalúa un espacio; de lo contrario, devuelve falso (número entero cero)	<code>isspace(' ')</code>
<code>int toupper(charExp)</code>	Devuelve el equivalente en mayúscula si <code>charExp</code> evalúa un carácter en minúscula; de lo contrario, devuelve el código de carácter sin modificación	<code>toupper('a')</code>
<code>int tolower(charExp)</code>	Devuelve el equivalente en minúscula si <code>charExp</code> evalúa un carácter en mayúscula; de lo contrario, devuelve el código de carácter sin modificación	<code>tolower('A')</code>

Nota:
Libro de Savitch
– Resumen de
funciones
(desde pág. 905)

LEER

Capítulo 6 (El tipo string)
Libro: Benjumea-Roldan
Universidad de Málaga

Capítulo 7 (7.2 La clase string)
Libro: “C++ para ingeniería y ciencias” 2da edición -
Gary J. Bronson