

Algoritmos y

Estructuras

De

Datos

ESTRUCTURAS

DE DATOS:

Cadenas de Caracteres /

Strings

ESTRUCTURAS DE DATOS

- **Simples o básicos:** caracteres, reales, flotantes.
- **Estructurados:** colección de valores relacionados. Se caracterizan por el tipo de dato de sus elementos, la forma de almacenamiento y la forma de acceso.

- **Estructuras estáticas:** Su tamaño en memoria se mantiene inalterable durante la ejecución del programa, y ocupan posiciones fijas.

ARREGLOS - CADENAS - ESTRUCTURAS

- **Estructuras dinámicas:** Su tamaño varía durante el programa y no ocupan posiciones fijas.

LISTAS - PILAS - COLAS - ARBOLES - GRAFOS

ESTRUCTURAS DE DATOS: Clasificaciones

- Según donde se almacenan
 - Internas** (en memoria principal)
 - Externas** (en memoria auxiliar)
- Según tipos de datos de sus componentes
 - Homogéneas** (todas del mismo tipo)
 - No homogéneas** (pueden ser de distinto tipo)
- Según la implementación
 - Provistas por los lenguajes** (básicas)
 - Abstractas** (TDA - Tipo de dato abstracto que puede implementarse de diferentes formas)
- Según la forma de almacenamiento
 - Estáticas** (ocupan posiciones fijas y su tamaño nunca varía durante todo el módulo)
 - Dinámicas** (su tamaño varía durante el módulo y sus posiciones también)

CARACTERES Y CADENAS: Fundamentos

□ Caracteres:

■ **Constante de carácter:**

- Un valor int representado como un caracter entre comillas simples
- Ejemplo: 'z' representa el valor entero de z

□ Cadenas:

■ **Series de caracteres tratados como una unidad**

- Pueden incluir letras, dígitos y caracteres especiales (*, /, \$)
- Representadas entre comillas dobles, por ejemplo "Hola"
- Terminan siempre en un carácter nulo ('\0')
- Las cadenas pueden ser llamadas también *strings*.

Cadenas de Caracteres

En C, una **cadena** es un tipo de dato compuesto, un **arreglo de caracteres (*char*)** que siempre incluye un 0 binario (frecuentemente llamado ***terminador nulo* – ‘\0’**), como elemento final del arreglo.

A	d	i	ó	s	\0					
0	1	2	3	4	5	6	7	8	9	10

Cadenas de Caracteres en C++

Existen dos alternativas para el manejo de cadenas:

- Cadenas al estilo de C (*terminadas en nulo*)
- Tipo string

Cadenas al estilo de C:

- ✓ Arrays de tipo char con una longitud máxima
- ✓ Un último carácter especial al final: '\0'

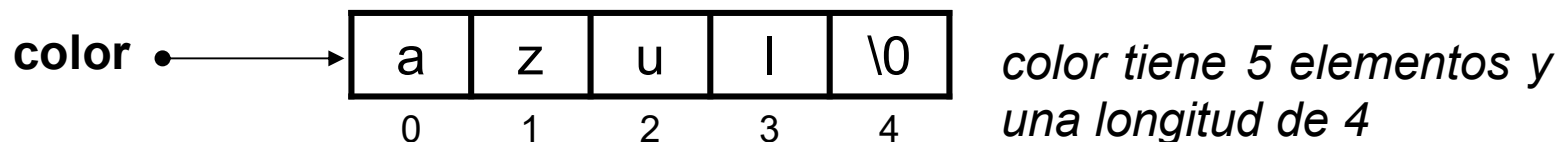
Tipo string:

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

Declaración de Cadenas

- Se declaran como un **arreglo de caracteres**
 - `char color[] = "azul";`
 - `char color[5] = "azul";`
- También se pueden declarar como una variable de tipo **char ***
 - `char *colorPtr = "azul";`
- El tamaño de la cadena debe incluir un byte más para poder almacenar el carácter `'\0'`
- Son un puntero al primer caracter

Puntero a un carácter o a una cadena



El **número total de caracteres** de una cadena es siempre igual a la longitud de la cadena más 1.

Inicialización de Cadenas

- El valor inicial de una cadena solamente se puede asignar en la línea de declaración:

- `char color[5] = "azul";`

- De otra manera, sería incorrecto:

- `char color[5];`
 - `color = "azul";`



Inicialización de Cadenas

- **No se deben confundir** las siguientes inicializaciones:
 - `char color[] = "azul";`
 - `char color[] = {'a', 'z', 'u', 'l'};`
- Estas dos inicializaciones *no son equivalentes*.
- La primera **coloca el carácter nulo '\0'** en el arreglo después de los caracteres 'a', 'z', 'u' y 'l'.
 - Esta forma de inicialización *coloca de manera automática* el carácter nulo '\0' al final del arreglo.
- La segunda **no coloca el '\0' en ninguna parte del arreglo**.
 - De esta manera, luego no será posible utilizar las funciones y librerías para manejo de cadenas.

Entrada de Cadenas

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.**

Por ejemplo, si como entrada a un programa introducimos por teclado la secuencia de caracteres **La Paz**, veremos que en realidad la cadena leída es **La**, ya que el carácter en blanco actúa como delimitador y fuerza el fin de la lectura.

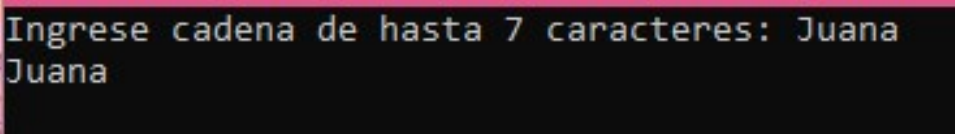
Entrada de Cadenas

- Copia lo ingresado por teclado hasta que se presione Enter
- Se debe dejar lugar en el arreglo para '\0'

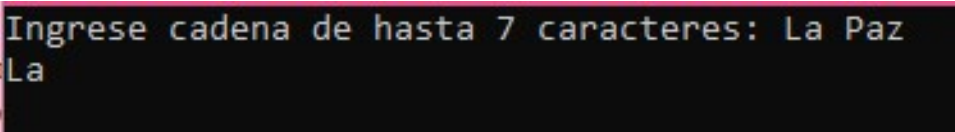
```
#include <iostream>
#define K 8
using namespace std;

int main() {
    char cadena[K];

    cout << "Ingrese cadena de hasta 7 caracteres: ";
    cin >> cadena;
    cout << cadena << endl;
    return 0;
}
```



Ingrese cadena de hasta 7 caracteres: Juana
Juana



Ingrese cadena de hasta 7 caracteres: La Paz
La

Si bien no se puede utilizar cout para mostrar los elementos de un arreglo numérico, sí se lo puede utilizar para mostrar los de una cadena.

Salida de cadenas

- Muestra por pantalla el contenido de una cadena
- También muestra el contenido de la cadena carácter a carácter

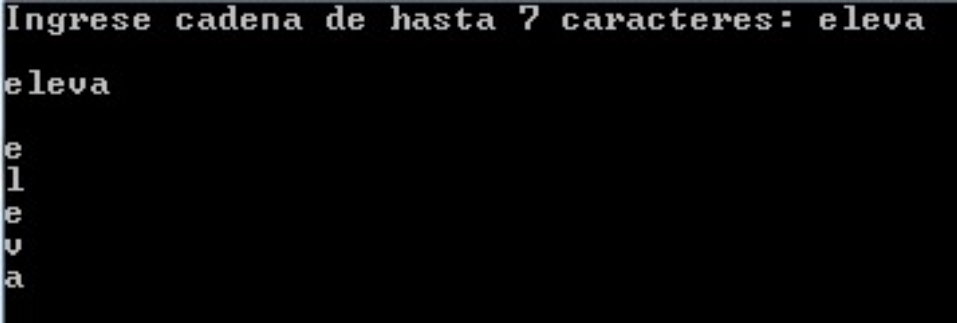
```
#include <iostream>
#define K 8
using namespace std;

int main() {
    int i;
    char cadena[K];

    cout << "Ingrese cadena de hasta 7 caracteres: ";
    cin >> cadena;

    cout << endl << cadena << endl << endl;

    i = 0;
    while (cadena[i] != '\0') {
        cout << cadena[i] << endl;
        i++;
    }
    return 0;
}
```



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

E/S de Caracteres y Cadenas

- Funciones en biblioteca <stdio>
- Permiten manejar caracteres y cadenas.

Prototipo de Función	Descripción
<code>int getchar(void);</code>	Lee el próximo caracter desde la entrada estándar y lo retorna como un entero.
<code>char *gets(char *s);</code>	Lee caracteres desde la entrada estándar en el arreglo S hasta un caracter de newline o end-of-file. Se agrega al array un caracter de terminación NULL.
<code>int putchar(int c);</code>	Imprime el caracter almacenado en c.
<code>int puts(const char *s);</code>	Imprime el string s seguido por un caracter de nueva línea.

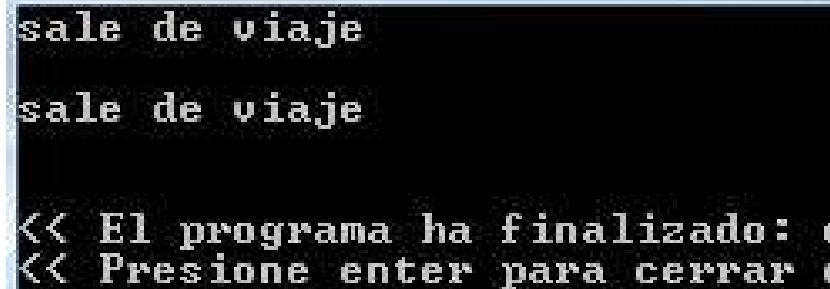
Entrada de Cadenas

- Lectura de cadenas con espacios
- **gets()** es la forma más fácil de leer una cadena con espacios.

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

#define K 15

int main() {
    char cadena[K];
    gets(cadena);
    cout << endl << cadena << endl;
    return 0;
}
```



```
sale de viaje
sale de viaje

<< El programa ha finalizado:
<< Presione enter para cerrar
```

gets() usa la biblioteca **cstdio**

E/S de Caracteres y Cadenas

- `getchar()` permite leer solo un caracter

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

int main() {
    char c;
    char cadena[80];
    int i = 0;

    puts("Ingrese una linea de texto:");
    while((c=getchar()) != '\n') {
        cadena[i] = c;
        i++;
    }
    cadena[i] = '\0';

    puts("\n La linea ingresada fue:");
    puts(cadena);

    return 0;
}
```

Tiene un comportamiento similar
a la lectura implementada con
`gets()`

```
Ingrese una linea de texto:
esta es una prueba


La linea ingresada fue:
esta es una prueba

<< El programa ha finalizado
<< Presione enter para cerrar
```


Librería de Manejo de Cadenas

cstring :

Operaciones disponibles para trabajar sobre cadenas que tienen el carácter nulo '\0'.

<div>  CUADRO 11.1 Algunas funciones de cadena tipo C predefinidas en <code><cstring></code> (parte 1 de 2) </div>		
Función	Descripción	Precauciones
<code>strcpy(Var_cadena_destino, Cadena_origen)</code>	Copia el valor de cadena tipo C <i>Cadena_origen</i> hacia la variable de cadena tipo C <i>Var_cadena_destino</i> .	No verifica que <i>Var_cadena_destino</i> sea lo bastante grande como para almacenar el valor de <i>Cadena_origen</i> .
<code>strncpy(Var_cadena_destino, Cadena_origen, Límite)</code>	Igual que la función <code>strcpy</code> de dos argumentos, sólo que se copian cuando mucho <i>Límite</i> caracteres.	Si <i>Límite</i> se elige con cuidado, esta función es más segura que la versión de dos argumentos de <code>strcpy</code> . No se implementa en todas las versiones de C++.
<code>strcat(Var_cadena_destino, Cadena_origen)</code>	Concatena el valor de cadena tipo C <i>Cadena_origen</i> con el final de la cadena tipo C que se encuentra en la variable de cadena tipo C <i>Var_cadena_destino</i> .	No verifica que <i>Var_cadena_destino</i> sea lo bastante grande como para almacenar el resultado de la concatenación.
<code>strncat(Var_cadena_destino, Cadena_origen, Límite)</code>	Igual que la función <code>strcat</code> de dos argumentos, sólo que se anexan cuando mucho <i>Límite</i> caracteres.	Si <i>Límite</i> se elige con cuidado, esta función es más segura que la versión de dos argumentos de <code>strcat</code> . No se implementa en todas las versiones de C++.
<code>strlen(Cadena_origen)</code>	Devuelve un entero igual a la longitud de <i>Cadena_origen</i> . (El carácter nulo '\0' no se cuenta en la longitud.)	

Copiar: strcpy y strncpy

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

```
int main()
```

```
{
```

```
    char x[] = "Algoritmos y Estructuras";
```

```
    char y[25] = "abc";
```

```
    char z[15] = "zzz";
```

```
    cout << "La cadena x contiene: " << x << endl;
```

```
    cout << "La cadena y contiene: " << y << endl;
```

```
    cout << "La cadena z contiene: " << z << endl;
```

```
    cout << "Copiado en y: " << strcpy(y,x) << endl;
```

```
    strncpy(z,x,10);
```

```
    cout << "Copiado 10 en z: " << z << endl;
```

```
    cout << endl << z[9] << z[10];
```

```
    return 0;
```

```
}
```

```
La cadena x contiene: Algoritmos y Estructuras
La cadena y contiene: abc
La cadena z contiene: zzz
Copiado en y: Algoritmos y Estructuras
Copiado 10 en z: Algoritmos
s
<< El programa ha finalizado: código de salida:
```

Concatenar: strcat y strncat

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

```
int main()
```

```
{
```

```
    char x[20] = "Feliz ";
    char y[] = "Año nuevo ";
    char z[20] = "";
```

```
    cout << "La cadena x contiene: " << x << endl << endl;
    cout << "La cadena y contiene: " << y << endl << endl;
    cout << "La cadena z contiene: " << z << endl << endl << endl;
```

```
    cout << "Concatenacion de y en x: " << strcat(x,y) << endl << endl;
```

```
    cout << "Asi quedo la cadena x: " << x << endl << endl;
```

```
    cout << "Concatenacion de 4 x en z: " << strncat(z,x,4) << endl << endl;
    cout << "Concatenacion de x en z: " << strcat(z,x) << endl << endl;
    return 0;
```

```
}
```

La cadena x contiene: Feliz

La cadena y contiene: Año nuevo

La cadena z contiene:

Concatenacion de y en x: Feliz Año nuevo

Asi quedo la cadena x: Feliz Año nuevo

Concatenacion de 4 x en z: Feli

Concatenacion de x en z: FeliFeliz Año nuevo

Longitud de una Cadena: strlen

- **strlen()** retorna el número de caracteres que contiene una cadena (*sin tener en cuenta* el carácter de fin de cadena)

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

int main(){
    char x[40] = "Esta es una sentencia con 7 tokens";

    cout << "La cadena: " << x
        << endl << "tiene una longitud de: "
        << strlen(x) << endl;

    return 0;
}
```

```
La cadena: Esta es una sentencia con 7 tokens
tiene una longitud de: 35

<< El programa ha finalizado: codigo de salida: 0 >>
<< Presione enter para cerrar esta ventana >>_
```

Comparación de cadenas

▶ CUADRO 11.1 Algunas funciones de cadena tipo C predefinidas en <code><cstring></code> (parte 2 de 2)		
Función	Descripción	Precauciones
<code>strcmp(Cadena_1, Cadena_2)</code>	Devuelve 0 si <i>Cadena_1</i> y <i>Cadena_2</i> son iguales. Devuelve un valor < 0 si <i>Cadena_1</i> es menor que <i>Cadena_2</i> . Devuelve un valor > 0 si <i>Cadena_1</i> es mayor que <i>Cadena_2</i> (es decir, devuelve un valor distinto de cero si <i>Cadena_1</i> y <i>Cadena_2</i> son distintas.) El orden es lexicográfico.	Si <i>Cadena_1</i> es igual a <i>Cadena_2</i> esta función devuelve 0, lo que la convierte en <i>falsa</i> . Esto es lo inverso de lo que podríamos esperar que devolviera si las cadenas son iguales.
<code>strncmp(Cadena_1, Cadena_2, Límite)</code>	Igual que la función <code>strcmp</code> de dos argumentos, sólo que se comparan cuando mucho <i>Límite</i> caracteres.	Si <i>Límite</i> se elige con cuidado, esta función es más segura que la versión de dos argumentos de <code>strcmp</code> . No se implementa en todas las versiones de C++.

Comparación de cadenas

- **int strcmp(cadena1, cadena2)**
 - Compara la cadena de caracteres *cadena1* con el *cadena2*
 - Retorna un:
 - **Número negativo**, si $\text{cadena1} < \text{cadena2}$,
 - **Cero**, si $\text{cadena1} == \text{cadena2}$ ó
 - **Número positivo**, si $\text{cadena1} > \text{cadena2}$
- **int strncmp(cadena1, cadena2, n)**
 - Compara n caracteres de las cadenas *cadena1* y *cadena2*
 - Retorna un valor de la misma forma que strcmp

La comparación se realiza comparando la representación numérica de los caracteres tal como están especificados en el orden del conjunto de caracteres que está siendo usado.

strcmp y strncmp

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

int main() {
    char x[20] = "Feliz ";
    char y[] = "Año nuevo ";
    char z[20] = "Felices vacaciones";

    cout << "La cadena x contiene: " << x << endl << endl;
    cout << "La cadena y contiene: " << y << endl << endl;
    cout << "La cadena z contiene: " << z << endl << endl;

    cout << "Comparo x con y: " << strcmp(x,y) << endl << endl;
    cout << "Comparo x con z: " << strcmp(x,z) << endl << endl;
    cout << "Comparo z con y: " << strcmp(z,y) << endl << endl;

    cout << "Comparo x con z: " << strncmp(x,z,6) << endl << endl;
    cout << "Comparo x con z: " << strncmp(x,z,7) << endl << endl;
    cout << "Comparo z con x: " << strncmp(z,x,7) << endl << endl;

    return 0;
}
```

```
La cadena x contiene: Feliz
La cadena y contiene: Año nuevo
La cadena z contiene: Felices vacaciones
Comparo x con y: 1
Comparo x con z: 1
Comparo z con y: 1
Comparo x con z: 23
Comparo x con z: 23
Comparo z con x: -23
```

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>

Funciones de Conversión de Cadenas

- **tolower()** permite convertir un caracter en minúscula

```
#include <iostream>
#include <cstdio>
#include <cctype>
#include <cstring>
using namespace std;

#define K 50
```

```
int main() {
    char cadena[K];

    puts("Ingrese una cadena toda en MAYUSCULAS");

    gets(cadena);
    for (int i=0; i<strlen(cadena); i++)
        cout << char(tolower(cadena[i]));

    return 0;
}
```

```
Ingrese una cadena toda en MAYUSCULAS
ESTA ES UNA PRUEBA
esta es una prueba
```

```
<< El programa ha finalizado: codigo de
```

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.

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

- ❑ Aunque, **no es posible emplear 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

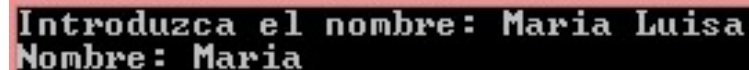
La entrada/salida de datos de tipo string sigue el **mismo esquema que la entrada/salida de los tipos predefinidos simples**.

Se basa en el uso de los operadores **>>** y **<<** sobre los flujos **cin** y **cout**.

La utilización del operador << sobre un flujo de salida cout muestra todos los caracteres que forman parte de la cadena.

```
#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
```

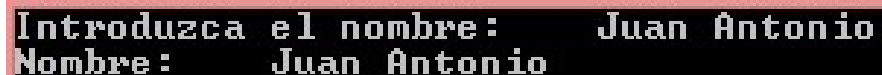
Tipo String: Entrada de Cadenas

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), sin eliminar los 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**.

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;  
}
```

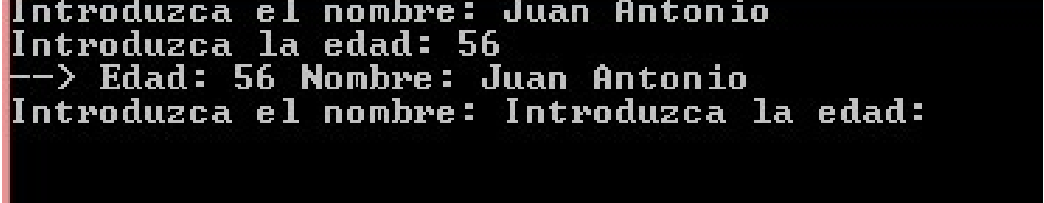
```
Introduzca el nombre: Juan Anto.nio  
Nombre: Juan Anto
```

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;
}
```



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 >>.

En la siguiente iteración, la función getline lee una secuencia de caracteres hasta encontrar un ENTER (sin saltar los espacios iniciales), por lo que 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 necesidad de detener 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, por lo que no será posible leer una cadena de caracteres vacía.

```
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 linea
        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()*

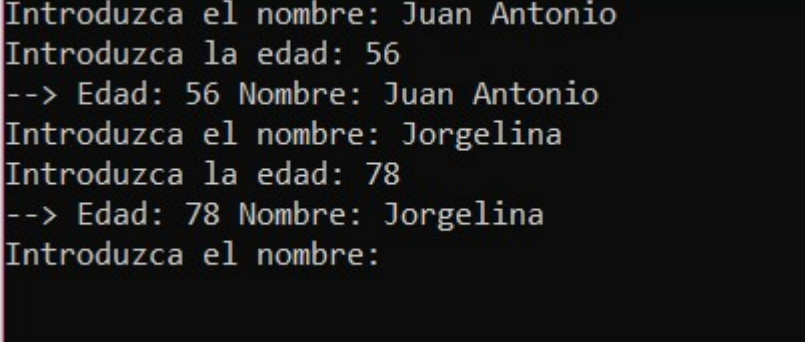
Es posible que interese que la **cadena vacía** sea una entrada válida en el programa.

En este caso, es necesario que el buffer se encuentre vacío en el momento de realizar la operación de entrada. Para ello, eliminaremos los caracteres que pudiera contener el buffer (no únicamente espacios en blanco) después de la última operación de lectura de datos, usando 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  $i \in [0..nombre.size()-1]$   
nombre[i] = 'z'; donde  $i \in [0..nombre.size()-1]$ 
```

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

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.

```
terminate called after throwing an instance of 'std::out_of_range'  
what():  basic_string::at  
  
This application has requested the Runtime to terminate it in an unusual way.  
Please contact the application's support team for more information.
```

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):** permite intercambiar *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"
```


LEER

**Página 531 (Cadenas de caracteres)
Libro: “Fundamentos de la programación” - Luis
Hernández Yáñez - Universidad Complutense**

**Página 91 (Cadenas)
Libro: “Estructuras de datos en C++” - Luis Joyanes**

**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**