

ElUniversidad Autónoma de Entre Ríos  
**Facultad de Ciencia y Tecnología**  
Sede: Oro Verde



---

# FUNDAMENTOS DE PROGRAMACIÓN

RESUMEN DE CONTENIDOS N°9  
**Cadenas de Caracteres**

## RESUMEN DE CONTENIDOS N° 9

# Cadenas de Caracteres

### Cadena de Caracteres

Es frecuente encontrarse con la necesidad de usar cadenas de caracteres. Por ejemplo, si se quieren guardar los datos de una persona ¿cómo se representaría su nombre, su apellido, su dirección, etc.?

En general los lenguajes de programación presentan dos posibilidades:

- a) ofrecer un tipo Cadena como tipo predefinido en el lenguaje, ó
- b) utilizar un caso especial de arreglo de caracteres, mediante una estructura array cuyo tipo base sea el tipo Char.

En C++ el tratamiento de las cadenas de caracteres se puede realizar de estas dos formas diferentes: utilizando el tipo **string** proporcionado por la biblioteca estándar, o implementado sobre array de caracteres.

### El tipo de dato **string**

En C++ el tratamiento de las cadenas de caracteres se puede lograr a través del tipo **string** que forma parte de la biblioteca estándar.

Un valor del tipo **string** será una secuencia de caracteres de longitud indefinida. Para poder utilizar el tipo **string**, al principio del programa se debe incluir

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

### Constantes y Variables de tipo String

Las constantes de tipo **string** se representan entre comillas, por ejemplo: "Juan Pérez". Se podrán definir tanto variables como constantes de dicho tipo, de la siguiente forma:

```
const string nombre = "María González"; //constante nombre de tipo string
string cadena;                          //variable cadena de tipo string, sin valor inicial
string apellido = "Pérez";              //variable apellido de tipo string con valor "Pérez"
string apel ("Pérez");                   // variable apel de tipo string con valor "Pérez"
string otro = nombre;                    // variable otro de tipo string con el valor de la
                                         // constante string nombre
```



**Nota:** 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 variables tipo String

El operador de entrada **cin>>** salta los espacios en blanco (y saltos de línea). Por lo tanto, al ingresar una cadena de caracteres, se leerá hasta encontrar un espacio en blanco o un salto de línea.

Ejemplo:

```
cin>>nombre;           // si se ingresara "Ana María", en nombre se guardaría sólo "Ana"
```

Esto es así, dado que el operador **>>**, elimina los espacios en blanco que hubiera al principio de la entrada de datos, y lee dicha entrada hasta que encuentre algún carácter de separación (ejemplo, el espacio en blanco).

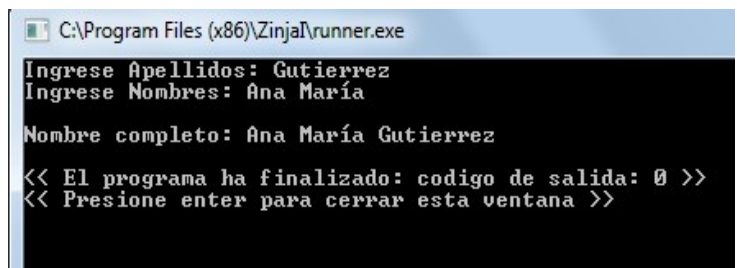
Por lo expuesto, no es posible utilizar el operador **>>** para leer una cadena de caracteres que incluya algún carácter en blanco.

Para poder ingresar el valor *"Ana María"* a la variable **nombre**, se debe utilizar la función **getline**, en lugar de **cin>>**, de la siguiente forma:

```
string cadena;  
getline(cin, cadena);           // lee una línea completa.
```

La función **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 y/o intermedios. Así pues, se podrían leer nombres compuestos utilizando **getline** de la siguiente forma:

```
#include <iostream>  
#include <string>  
using namespace std ;  
int main()  
{  
    string apellidos, nombres;  
    cout<<"Ingrese Apellidos: ";  
    getline(cin,apellidos);  
    cout<<"Ingrese Nombres: ";  
    getline(cin,nombres);  
    cout<<endl; //deja un renglón en blanco  
    cout<<"Nombre completo: "<<nombres<<" "<<apellidos;  
    return 0;  
}
```



```
C:\Program Files (x86)\Zinja\runner.exe  
Ingrese Apellidos: Gutierrez  
Ingrese Nombres: Ana María  
  
Nombre completo: Ana María Gutierrez  
  
<< El programa ha finalizado: codigo de salida: 0 >>  
<< Presione enter para cerrar esta ventana >>
```

Además, la función **getline** permite especificar el delimitador que marca el final de la secuencia de caracteres a leer. Si no se especifica ninguno (como ocurrió en el ejemplo anterior), se utiliza por defecto el carácter de fin de línea (**ENTER**). Sin embargo, si se especifica el delimitador, la función **getline** 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.

```
getline(cin, cadena, ' '); // lee hasta encontrar ' '.
```

Véase el siguiente ejemplo:

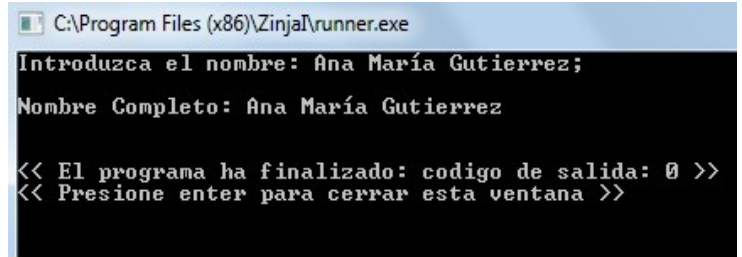
```
#include <iostream>
```

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

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

```
int main()
```

```
{
    string nomyape;
    cout<<"Introduzca el nombre: ";
    getline(cin, nomyape, DELIMITADOR);
    cout<<endl; //deja un renglón en blanco
    cout<<"Nombre Completo: "<<nomyape<< endl;
    return 0;
}
```



Como hemos visto, el comportamiento de las operaciones de lectura con `>>` y **getline** es diferente. En ocasiones, cuando se utiliza una lectura con **getline** después de una lectura previa con `>>`, puede encontrarse con un comportamiento que, aunque correcto, puede no corresponder al esperado intuitivamente. Conviene conocer con detalle qué ocurre en cada caso, por lo que a continuación se verá sobre un ejemplo.

Supongamos que se quiere diseñar un programa que solicite y muestre: nombre y edad de dos personas. Para ello, se leerá el nombre de la persona con **getline** (ya que puede ser compuesto), y la edad con el operador de entrada `>>`, ya que permite introducir datos numéricos:

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

```
int main()
```

```
{
    string nomyape1, nomyape2;
    int edad1, edad2;

    cout<<"Introduzca el nombre de la primer persona: ";
    getline(cin, nomyape1);
    cout<<"Introduzca edad de la primer persona: ";
    cin>>edad1;
    cout<<"Introduzca el nombre de la segunda persona: ";
    getline(cin, nomyape2);
    cout<<"Introduzca edad de la segunda persona: ";
    cin>>edad2;
    cout<<endl;
    cout<<"Edad: "<<edad1<<" Nombre Completo: "<<nomyape1<<endl;
    cout<<"Edad: "<<edad2<<" Nombre Completo: "<<nomyape2<<endl;
    return 0;
}
```

Sin embargo, al ejecutar el programa se comprueba que no funciona como se esperaba. La primera iteración funciona adecuadamente, el flujo de ejecución espera hasta que se

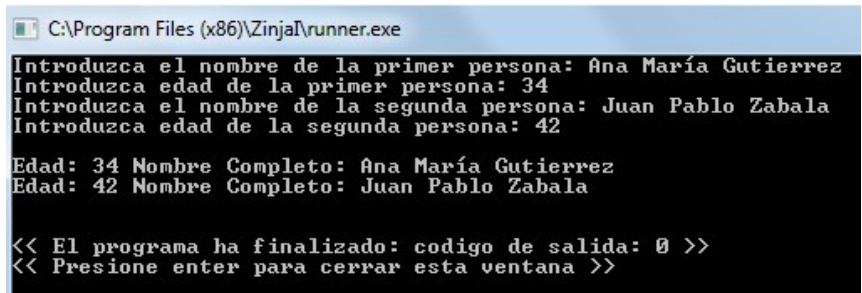
introduzca el nombre, y sin embargo, la siguiente carga, omite o saltea el ingreso del nombre, y habilita la carga de la edad. Es decir, el programa no se detiene para que el usuario pueda introducir el nombre.

Esto es debido a que no se ha tenido en cuenta cómo se comportan las operaciones de lectura de datos (>> y **getline**) al obtener los datos del buffer de entrada. Debe considerarse que después de leer la primer edad, 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 carga, 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. 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.

Para este ejemplo, utilizaremos la función **cin.get()**, la cual lee un carácter sin eliminar espacios en blanco iniciales.

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



```
int main()
{
    string nomyape1, nomyape2;
    int edad1,edad2;

    cout<<"Introduzca el nombre de la primer persona: ";
    getline(cin,nomyape1);
    cout<<"Introduzca edad de la primer persona: ";
    cin>>edad1;
    cin.get();
    cout<<"Introduzca el nombre de la segunda persona: ";
    getline(cin,nomyape2);
    cout<<"Introduzca edad de la segunda persona: ";
    cin>>edad2;
    cout<<endl;
    cout<<"Edad: "<<edad1<<" Nombre Completo: "<<nomyape1<<endl;
    cout<<"Edad: "<<edad2<<" Nombre Completo: "<<nomyape2<<endl;
    return 0;
}
```

También es posible eliminar un número determinado de caracteres del flujo de entrada, o hasta que se encuentre un determinado carácter:

```
{
    cin.ignore(); // elimina el próximo carácter

    cin.ignore(5); // elimina los 5 próximos caracteres
```

```
        cin.ignore(1000, '\n'); // elimina 1000 caracteres o hasta ENTER
    (nueva-línea)
}
```

Si bien, en los ejemplos vistos ya se ha utilizado, es momento de destacar que para mostrar el contenido de una variable **string**, se utiliza **cout<<**, de la misma manera que en el caso de variables de otros tipos.

## Operadores y operaciones con variables tipo String

- **Asignación**

A una variable de tipo **string** se le puede asignar directamente tanto una cadena de caracteres constante como otra variable de tipo **string** (o de tipo caracter).

La asignación se realiza con el signo = (igual).

```
const string nombre = "María";
string cadena;

cadena= nombre;           // asigna el valor de nombre a cadena
```

- **Concatenación**

Se puede crear una cadena formada por varias cadenas, concatenándolas (sumándolas).

La concatenación de cadenas se realiza a través del signo + (más).

En la concatenación, al menos uno de los dos primeros operandos debe estar declarado de tipo **string**.

La operación de concatenación será imprescindible para añadir caracteres a una cadena porque no es posible acceder fuera del rango de la misma.

```
string texto1, texto2 = "Hola", texto3(" Que tal");
texto1 = texto2 + texto3 + " estás? "; //texto1 será: "Hola Que tal estás?"
```

```
string cadena = "José"
cadena = cadena + ' ';           //añade un espacio al final de cadena
cadena = cadena + "García";      //cadena será "José García"
```

```
string nom;
nom = "juan"+ "Luis";           //Error. Al menos uno de los operandos debe estar declarado como string
```

- **Subcadenas**

Es posible generar una subcadena o una nueva cadena a partir de una cadena de caracteres, de la siguiente forma:

```
string cadena1="Hola Que tal";  
string subcadena1(cadena1, 5, 7); //subcadena1 será: "Que tal". 7 letras de cadena1 desde la  
// sexta. El primer carácter ocupa la posición 0.
```

Otra forma de lograr subcadenas, es mediante la operación **substr**, la cual tiene el siguiente formato:

```
cad. substr(i, t);
```

Donde:

Cad: es una cadena de caracteres *string*

i : posición inicial

t : cantidad de caracteres de t.

Ejemplo:

```
string subcadena2;  
subcadena2 = cadena1.substr(0,4) //subcadena2 será: "Hola ". 4 letras de cadena1, desde el  
//comienzo.
```

- **Insertar un texto en una cadena**

```
string cadena1 = "Hola Que Tal"  
cadena1.insert(5, "Juan "); //Inserta un texto en cadena1 en la posición 6. El resultado será:  
// "Hola Juan Que tal"
```

- **Reemplazar letras en una cadena**

```
string cadena1 = "Hola Que Tal"  
cadena1.replace(1, 2, "ad"); //cambia dos letras a partir de la posición 2. El resultado será:  
// "Hada Que tal"
```

- **Longitud de una cadena**

Para saber el número de caracteres de una cadena (**string**), se utiliza **size()**:

```
string cadena1 = "Hola Que Tal"  
int n = cadena1.size(); // n tendrá el valor 12
```

- **Acceso a un carácter de la cadena**

Se accede a los caracteres que componen una cadena mediante indexación, donde el primer elemento se encuentra en el índice 0, y el último elemento en el índice `size() - 1`:

```
string cadena = "juan antonio";  
char primera_letra = cadena[0]; // primera_letra tendrá el valor j  
char ultima_letra = cadena[cadena.size()-1]; // última_letra tendrá el valor o  
cadena[0] = 'J' // pone el primer carácter de la cadena en J (mayúscula)
```

- **Comparación de cadenas**

Es posible comparar variables y constantes de tipo **string** mediante los operadores relacionales (==, !=, >, >=, <, <=) siempre y cuando al menos uno de los operandos esté declarado como de tipo string.

Ejemplos:

```
string nombre = "Jose";
string cadena = 'Jose Pérez';
if (nombre == "Jose"){...} if (nombre != "Jose"){...}
....
if (cadena > nombre){...}
.....
If ("juan"== "juan"){...} //Error. Al menos uno de los operandos debe estar
// declarado como string
```

- **Buscar una cadena dentro de otra**

Es posible buscar si una cadena se encuentra dentro de otra cadena, de la siguiente forma:

```
string cadena1 = "Hola Que Tal"
int n=cadena1.find("Que"); //Busca la subcadena "Que" dentro cadena1. n valdrá 5 que es la
//posición donde empieza la cadena buscada
```

Si la cadena no se encuentra, find() devolverá -1,

Otra forma de usar find es:

```
string cadena1 = "Hola Que Tal"
int PosInicial = 7
int n=cadena1.find("Que",PosInicial); //Busca la subcadena "Que" dentro cadena1, a
// partir de la posición 7. n valdrá -1.
```

- **Convertir una cadena en un valor numérico**

A veces es necesario transformar el valor de una cadena, en un valor numérico. Se utilizarán las funciones de c++:

- stoi() para transformar una cadena a un valor entero
- stol() para transformar una cadena a un valor entero largo
- stof() para transformar una cadena a un valor real.

Ejemplos:

```
string EdadStr="12";
string CUILStr="204125679";
string SueldoStr="3456.98";

int edad = stoi( EdadStr ); // edad valdrá 12.
long ciul = stol( CUILStr ); // cuil valdrá 204125679.
float sueldo = stof( SueldoStr ); // sueldo valdrá 3456,98.
```