

Metodología de la Programación

Tema 0.- Tipos de datos

- Tipos de datos en C++**
- Conversiones de tipos**
- Otros tipos de datos**
- Operadores**
- Cadenas de tipo string**



ugr Universidad de Granada DECSAI Universidad de Granada

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons] 1

Tipos de datos en C++

Tipos aritméticos

Tipos integrales	<code>bool</code> <code>char</code> <code>int</code>	<code>signed</code> <code>unsigned</code> <code>signed</code> <code>unsigned</code>	<i>Modificadores de signo</i> <i>Modificadores de tamaño</i>
	<code>short</code> <code>long</code>		
Coma flotante	<code>float</code> <code>double</code> <code>long double</code>	<i>Precisión simple</i> <i>Precisión doble</i> <i>Precisión extendida</i>	

Modificadores de tamaño: alteran el **número de bytes** que se usan para representar un dato.

Modificadores de signo: alteran el **rango** de valores representados.

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons] 2

Tipos de datos en C++

Tipos aritméticos - bool




El tipo **bool** permite la representación de valores de verdad: **true** **false**

```
bool vl; // Declaración
...
vl=false; // Asignación
...
if (vl==false) { ... } if (vl) { ... }
```

La conversión entre **bool** e **int** es:

```
false = 0 cout << vl ;
true = 1 cin >> vl;
```

*En C no existe bool
En su lugar se suele usar int
false ≡ 0
true ≡ cualquier otro valor*

```
int vl;
...
vl=0;
...
if (vl) { // Si vl!=0 ...
}
if (vl==0) { // Si false ...
}
```

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 3

Tipos de datos en C++

Tipos aritméticos - char




El tipo **char** permite almacenar **números enteros** (pequeños).
Eventualmente, es posible interpretar el número como el código ASCII de un *carácter*.

Ocupan 1 byte → podemos almacenar 2^8 números diferentes.

Declaración	$\left\{ \begin{array}{ll} \text{signed char} & -128..127 \\ \text{unsigned char} & 0..255 \\ \text{char} & \left\{ \begin{array}{l} 0..255 \\ -128..127 \end{array} \right. \end{array} \right\}$	Rango de valores que puede almacenar
-------------	--	--------------------------------------

```
signed char a=212; // 11010100
unsigned char b=212;
int x;

cout << a << b << endl;
x=a ; cout << x << endl;
x=b ; cout << x << endl;
```

cout y cin los interpretan como caracteres.

→ È È
-44
212

```
unsigned char a, b, c;
a = 10;
b = 20;
c = a + b; // c vale 30
```

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 4

Tipos de datos en C++

Tipos aritméticos - int

El tipo **int** permite almacenar **números enteros**.

El tipo puede ir precedido de un modificador de signo y otro de tamaño.

	Pequeño	Normal	Grande
Con signo	short int short signed short int signed short	int signed int signed	long int long signed long int signed long
Sin signo	unsigned short int unsigned short	unsigned int unsigned	unsigned long int unsigned long

El tamaño de cada tipo dependerá del sistema utilizado.

Ejemplo (Pentium IV + GNU/Linux + GNU g++):

- int** y **long** : 4 bytes [-2.147.483.648 , 2.147.483.647]
- unsigned int** : 4 bytes [0 , 4.294.967.295]
- short int** : 2 bytes [-32.768 , 32.767]

Tipos de datos en C++

Tipos aritméticos – float y double

Para representar **números reales** se usan:

{	float	<i>Precisión simple</i>
	double	<i>Precisión doble</i>
	long double	<i>Precisión extendida</i>

El tamaño concreto de cada uno depende del sistema.

Ejemplo (Pentium IV + GNU/Linux + GNU g++):

	float	double	long double
Mínimo	1.17549e-38	2.22507e-308	3.3621e-4932
Máximo	3.40282e+38	1.79769e+308	1.18973e+4932

... y negativos

Tipos de datos en C++

Constantes literales



Para escribir constantes literales de cada tipo aritmético se puede añadir un sufijo:

se escribe ...	el tipo es ...	ejemplos ...
entero	<code>int</code>	2456 -12345
enteroL	<code>long int</code>	45L -567L
enteroU	<code>unsigned int</code>	67U
enteroUL	<code>unsigned long int</code>	123456UL
real	<code>double</code>	1.23 -.46 2e4
realF	<code>float</code>	45.F .007F
realL	<code>long double</code>	67.56L -.56e-4L

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 7

Tipos de datos en C++

Tamaño de los tipos



C++ sólo garantiza el **tamaño mínimo** de cada tipo:

Tipo	$\left\{ \begin{array}{ll} \text{char} & 1 \\ \text{short int} & 2 \\ \text{int} & 2 \\ \text{long int} & 4 \\ \text{float} & 4 \\ \text{double} & 8 \\ \text{long double} & 8 \end{array} \right\}$	<i>Tamaño mínimo (bytes)</i>
------	--	------------------------------

El operador `sizeof` se puede usar para averiguar el tamaño de un objeto

<code>sizeof(float)</code>

<code>sizeof(56)</code>

<code>char *p;</code>
<code>sizeof(p)</code>

<code>char x;</code>
<code>sizeof(x)</code>

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 8

Tipos de datos en C++

Consulta de propiedades numéricas

```
#include <limits>
```

#include <climits>	#include <limits.h>
#include <cfloat>	#include <float.h>

Estos ficheros también tienen algunas de estas propiedades pero no son tan completos.

La forma de consultar algo sobre un tipo es:

```
numeric_limits<tipo>::propiedad_consultada
```

- Valores mínimo y máximo representables: `min()`, `max()`

```
cout << numeric_limits<int>::min() << endl;
cout << numeric_limits<float>::max() << endl;
```
- Si es o no signed: `is_signed`

```
cout << numeric_limits<char>::is_signed << endl;
```
- Epsilon (sólo reales): `epsilon()`
Devuelve la diferencia entre 1 y el menor valor mayor que 1.


```
cout << numeric_limits<double>::epsilon() << endl;
```

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 9

Metodología de la Programación

Tema 1.- Tipos de datos

- Tipos de datos en C++**
- Conversiones de tipos**
- Otros tipos de datos**
- Operadores**
- Cadenas de tipo string**



ugr Universidad de Granada DECSAI Universidad de Granada

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 10

Conversiones de tipos

Tipos de conversiones



Una buena norma de programación al construir expresiones es **operar**, entre sí, **datos de un mismo tipo** (evitar operar datos de tipos diferentes).

$2 + 2$ entero + entero = entero

Pero a veces es necesario operar datos de diferente tipo.

$2 + 2.4$ entero + real = ?

Los operadores están pensados para operar datos de un mismo tipo entre sí.

Cuando mezclamos distintos tipos: **se convierten los datos a un mismo tipo común y después se operan.**

Las conversiones pueden ser:

Implicitas	<i>Las hace el compilador de forma automática.</i>
Explicitas	<i>El programador indica que conversión quiere hacer.</i>

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons] 11

Conversiones de tipos

Conversiones implícitas



Las conversiones implícitas las hace el compilador de forma automática.

Promoción de tipos: Cuando opero datos numéricos, se convierten (temporalmente) los datos de menor capacidad al tipo de los de mayor capacidad.

$2 + 2.4 = ?$

Diagram illustrating type promotion:

```

    graph LR
      A["2  
entero"] -- "promoción" --> B["2.0  
real"]
      B -- "+" --> C["2.4  
real"]
      C -- "=" --> D["4.4  
real"]
  
```

Conversion hierarchy (from lowest to highest precision):

- long double
- double
- float
- unsigned long int
- long int
- unsigned int
- int
- unsigned short int
- short int
- unsigned char
- char
- bool

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons] 12

 **Conversiones de tipos**
Conversiones implícitas

A veces las conversiones pueden provocar la pérdida de datos.

```
int x;
x = 7.6;
```

Promoción inversa de tipos

Lo normal es que el compilador avise pero podría no ocurrir.

Las conversiones implícitas también se producen al establecer el paso de parámetros a funciones y en la devolución de valores.

```
unsigned int funcion(float f, double d)
{
    ...
    return 400; // 400 es una constante literal de tipo int
}
...
char c = funcion(7.13,56); // Llamada a la función
```

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada)  13

 **Conversiones de tipos**
Conversiones explícitas

El programador puede hacer conversiones de forma explícita (*casting* o *moldeados*).

(Tipo)expresión

Esto devuelve el resultado de evaluar la **expresión** y convertirla (promocionarla) al tipo dado por **Tipo**.

Esta es la sintaxis que se usa en C (también válida en C++):

En C++ disponemos de 4 operadores para hacer conversiones (no se recomienda usar la sintaxis de C anterior):

static_cast<Tipo>(Expresión)

reinterpret_cast<Tipo>(Expresión)

const_cast<Tipo>(Expresión)

dynamic_cast<Tipo>(Expresión)

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada)  14

Conversiones de tipos

Conversiones explícitas

static_cast<Tipo>(Expresión)

Permite realizar conversiones que se podrían considerar “con sentido”. De esta forma evitamos, por ejemplo, los avisos del compilador.

Ej: Conversiones entre coma flotante y tipos integrales.

```
int a;    float b;
a = static_cast<int>(b);
```

Tipo(expresión)

Aunque esto no se puede considerar una conversión explícita, el resultado para tipos intrínsecos es el mismo que el de un static_cast.

reinterpret_cast<Tipo>(Expresión)

Permite realizar conversiones entre tipos no relacionados entre sí. Por ejemplo entre distintos tipos de punteros.

Peligrosa y no recomendada salvo que se sepa lo que se hace.

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) | 15

Metodología de la Programación II

Tema 1.- Tipos de datos

- Tipos de datos en C++**
- Conversiones de tipos**
- Otros tipos de datos**
- Operadores**
- Cadenas de tipo string**



Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) | 16

Otros tipos de datos
El tipo enumerado

Ejemplo: Representa el día de la semana

```
int dia_semanal; // 0=Lunes, 1=Martes, ...
char dia_semana2; // 'L'=Lunes, 'M'=Martes, ...
```

Inconveniente: tenemos que recordar la codificación de cada día.

Un tipo enumerado es un tipo de dato creado por el programador en base al dominio de valores que puede tomar.

```
enum NuevoTipo { lista_valores };

enum DiaSemana { Lunes, Martes, Miercoles, Jueves,
                 Viernes, Sabado, Domingo };
...
DiaSemana d1, d2, d3=Domingo;
d1=Miercoles;
if (d2==Sabado) ...
```

Los valores concretos se asimilan a constantes con nombre.

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons] 17

Otros tipos de datos
El tipo enumerado

Podemos suponer que la forma de representar internamente un tipo enumerado es mediante un tipo `int` (aunque podría ser mediante algún otro tipo integral).

Cada elemento de la lista de valores se representa por 0, 1, 2, ...

```
enum DiaSemana { Lunes, Martes, Miercoles, Jueves,
                 Viernes, Sabado, Domingo };
```

La representación de cada dato sería:

```
0=Lunes, 1=Martes, 2=Miercoles, 3=Jueves, ...
```

Se podrían modificar esos valores por defecto:

```
enum DiaSemana { Lunes=20, Martes=10, Miercoles=33,
                 Jueves=46, Viernes, Sabado, Domingo };
```

Los que no se especifican se asignan de forma consecutiva desde el último valor usado:

```
47=Viernes, 48=Sabado, 49=Domingo
```

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons] 18

Otros tipos de datos

El tipo enumerado




Se pueden usar directamente como constantes enteras.

```
enum DiaSemana { Lunes, Martes, Miercoles, Jueves,
                 Viernes, Sabado, Domingo };
int x = Lunes+2; // x=0+2=Miercoles
```

Podemos ver el contenido usando `cout`:

```
cout << x;
```

En pantalla se muestra el número que representa a `x`.

No podemos usar `cin` para leer su valor ni asignarle ningún tipo integral:

```
cin >> x; // Error
x = 3; // Error
x = static_cast<DiaSemana>(3); // OK
x = static_cast<DiaSemana>(100); // Indefinido
```

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons] 19

Otros tipos de datos

La instrucción `typedef`




La instrucción `typedef` permite definir un sinónimo de un tipo de dato ya existente.

```
typedef TipoExistente NuevoTipo ;
```

```
typedef int Entero;
...
int x;
Entero y;
```

```
typedef unsigned char uchar;
typedef unsigned short int usint ;
...
uchar x;
usint y;
```

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons] 20

Metodología de la Programación

Tema 1.- Tipos de datos

- Tipos de datos en C++*
- Conversiones de tipos*
- Otros tipos de datos*
- Operadores*
- Cadenas de tipo string*



UGR Universidad de Granada DECSAI Universidad de Granada

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons] 21

Operadores

Operadores lógicos a nivel de bit

Hacen cálculos con los bits individuales de la representación.

Se pueden aplicar sobre tipos integrales y enumerados (bool, char, int, enum, ...)

&		^	-	>>	<<
Y	O	XOR	NOT	Despl. Derecha	Despl. Izquierda

```
unsigned char c = 37;      // c=00100101 (binario)
```

```
unsigned char r = c & 7;    // r=00000101
```

$$\begin{array}{r} 00100101 \\ \& 00000111 \\ \hline 00000101 \end{array}$$

```
unsigned char r = ~c;      // r=11011010
```

```
unsigned char r = c<<2;  // r=10010100
```

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons] 22

Operadores
Otros operadores




`+ = - = * = / = % = <=> = & = | = ^ =`

`var op= expresión` Forma de uso
... que equivale a esto: `var = var op expresión`

```
int x=7;

x += 4;      // x = x+4 = 11
x <= 2;      // x = x <= 2 = 0...001011<<2 = 0...0101100
```

`++ -- (post/pre incremento/decremento)`

`var++;` Forma de uso
`var--;` ... que equivale a esto `var = var+1;`
`var = var-1;`

```
int x=7;
x++;           // x = x+1 = 8
int y = (x++)*2;    // y=x*2; x++;      (y=16 x=9)
int z = (++x)*2;    // x++; z=x*2;      (z=20 x=10)
```

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons] 23

Operadores
Otros operadores




Expresión condicional

`expr1 ? expr2 : expr3`

Se evalúa `expr1` y:

- Si vale `true` se devuelve `expr2`.
- Si vale `false` se devuelve `expr3`.

```
x = (y>7) ? 9 : 6;      // x = 9 si (y>7)
// x = 6 si (y<=7)

x = (2*y<6*z) ? (sqrt(2.0*y)+3) : (z-1);
```

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons] 24

Operadores

Prioridad de los operadores

-> . [] () static_cast reinterpret_cast ++ -- (post)
sizeof ~ ! & new delete ++ -- (pre) + - (unario) * (desreferencia) () (casting tipo C)
.* (selección de miembro) ->* (selección de miembro)
* (multiplicación) /
+ (suma) - (resta)
<< >>
< <= > >=
== !=
&
^
&&
? :
= *= /= %= += -= <<= >>= &= ^= =
,

The diagram illustrates the precedence of C++ operators. It shows a vertical stack of operator symbols from highest to lowest precedence. A red curved arrow points from the top of the stack down to the bottom, indicating the flow of increasing precedence. The operators are grouped into categories: assignment (|=, *=, /=, %=, +=, -=), output (operator << and >>), comparison (==, !=, <, <=, >, >=), logical (operator && and ||), bitwise (operator ^), and other (operator ~, operator !, operator &, operator |, operator ?:). The stack ends with a comma (,).

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons] 25

Metodología de la Programación

Tema 1.- Tipos de datos

- Tipos de datos en C++**
- Conversiones de tipos**
- Otros tipos de datos**
- Operadores**
- Cadenas de tipo string**

ugr Universidad de Granada DECSAI Universidad de Granada

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons] 26

Cadenas de tipo string

¿Qué es un string?

string es una **clase** de la biblioteca estándar de C++ para la manipulación de cadenas de caracteres (es una alternativa a **char***).

Para usarla hay que incluir el fichero de cabecera:

```
#include <string>
```

Está dentro del espacio de nombres **std**:

```
using namespace std; std::string
```

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string c1="hola", c2;
    cout << c1 << endl;
    cout << "Escribe algo:";
    cin >> c2;
    cout << c2 << endl;
}
```

```
#include <iostream>
#include <string>
// NO hacemos using namespace std
int main()
{
    std::string c1="hola", c2;
    std::cout << c1 << std::endl;
    std::cout << "Escribe algo:";
    std::cin >> c2;
    std::cout << c2 << std::endl;
}
```

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) 27

Cadenas de tipo string

Declaración, inicialización y asignación

A un **string** le podemos asignar una cadena **char*** o **const char***:

```
string c1="hola";
string c2;
char *cc=new char[10];
strcpy(cc,"adiós");
...
c2 = cc;
```

Podemos crear un nuevo **string** a partir de otro o de un **char*** de la forma:

```
string c1("hola");
string c2(c1);
char *cc=new char[10];
strcpy(cc,"adiós");
...
string c3(cc);
```

A un **string** le podemos asignar otro **string**:

```
string c1,c2;
...
c2 = c1;
```

No hay que preocuparse por el tamaño del string.

La memoria se gestiona de forma automática.

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) 28

 **Cadenas de tipo string** 

Operadores básicos

Operadores básicos definidos sobre `string`:

- **Relacionales:** `==`, `!=`, `<`, `<=`, `>`, `>=`
Permiten comparar `string` con `string` y con `char*`
- **Concatenación:** `+`, `+=`
Permiten concatenar `string` con `string`, `char*` y `char`
- **Acceso a un carácter:** `[]`

```
string c1, c2="pepe";
...
if (c1==c2) cout << "Son iguales";
if (c1<"hola") cout << "c1 es menor que hola";
c1 = c2 + " " + "garcía";    // c1="pepe garcía"
c1 += " pérez";              // c1="pepe garcía pérez"
c1[0] = 'P';                  // c1="Pepe garcía pérez"
if (c1[5]=='g') cout << "La 6ª letra es g";
```

 **Cadenas de tipo string** 

Conversión de `string` a `char*`

Para convertir de `char*` a `string` basta con hacer una asignación:

```
char *cc="hola";
string c2;
c2 = cc;
```

A la inversa, no es posible usar el operador de asignación.

```
cc = c2; // ERROR
```

→ **c_str()**

Es una operación que devuelve un puntero al contenido del `string` pero en formato `char*`

```
string c1="pepe";
char *cc;
cc = c1.c_str();
```

Importante: el puntero devuelto por `c_str()` apunta a una zona de memoria contenida en el `string`.

 **Cadenas de tipo string** 

Otras operaciones

→ **Longitud de la cadena: `size()`, `length()`**
 Devuelven el número de caracteres de la cadena.

```
string c1="pepe";
cout << c1.length() << endl;      // Escribe 4
cout << c1.size() << endl;        // Idem
```

→ **Borrar la cadena (o una parte de ella): `clear()`, `erase()`**

```
string c1="abcdefgh";
c1.clear();                      // Lo borra todo (equivale a c1="")
cout << c1 << endl;              // Escribe ""
string c2="pepe";
c2.erase();                       // Lo borra todo
cout << c2 << endl;              // Escribe ""
string c3="abcdefgh";
c3.erase(3,2);                   // Borra 2 a partir del 4º carácter
cout << c3 << endl;              // Escribe "abcgfh"
string c4="abcdefgh";
c4.erase(3);                     // Borra del 4º carácter en adelante
cout << c4 << endl;              // Escribe "abc"
```

 **Cadenas de tipo string** 

Otras operaciones

→ **Búsqueda de subcadenas: `find()`, `rfind()`**
 Búsqueda una subcadena dentro de otra (hacia delante y hacia atrás)
 Devuelve la posición a partir de la cual ha sido encontrada
 (o la constante `string::npos` si no la encuentra).

```
string c1="abcdefdeg",c2="bc";

// Buscamos la subcadena "de" en la cadena c1
cout << c1.find("de") << endl;      // Escribe 3

// Idem pero la búsqueda empieza por el final de c1
cout << c1.rfind("de") << endl;      // Escribe 6
```



Cadenas de tipo string

Otras operaciones

→ Inserción de subcadenas: `insert()`

Inserta una subcadena dentro de una cadena.

```
string c1="abcdefgh",c2="xxx";
c1.insert(3,"zzz");
cout << c1 << endl;      // Escribe "abczzdefgh"
c1.insert(1,c2);
cout << c1 << endl;      // Escribe "axxxbczzdefgh"
```

→ Sustitución de un trozo de cadena por otro: `replace()`

Cambia un trozo de una cadena por una subcadena.

```
string c1="abcdefgh",c2="123456";
c1.replace(2,3,c2);
cout << c1 << endl;      // Escribe "ab123456fgh"
c1.replace(3,4,"X");
cout << c1 << endl;      // Escribe "ab1X6fgh"
```

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 33