Programación 1 **Tema 10**

Caracteres y cadenas de caracteres





Índice

- Caracteres
- Cadenas de caracteres
- Problemas



El tipo carácter

- Tipos carácter
 - Dominio de valores
 - Representación de los valores
 - □ Externa (en C++)
 - Interna (en la memoria del computador)
 - Operadores asociados



CaracteresDominio de valores



Fuente: Wikimedia Commons contributors, 'File:Brother typewriter by awdean1.jpg', *Wikimedia Commons, the free media repository,* 2016, https://commons.wikimedia.org/w/index.php?title=File:Brother typewriter by awdean1.jpg 4



Caracteres

Posible dominio de valores

- Letras mayúsculas del alfabeto inglés
- Letras minúsculas del alfabeto inglés
- Dígitos
- Signos de puntuación
- Signos matemáticos
- Letras con diacríticos (alfabetos latinos occidentales)
- Letras alfabetos centro-europeos
- Letras alfabeto griego
- Letras alfabeto cirílico
- Letras alfabetos asiáticos

CaracteresUnicode

- □ Estándar de codificación de caracteres
- Dominio de valores:
 - Alfabeto latino: Abcdefghijklmnopqrstuvwxyz
 - Alfabeto griego: α β γ δ ε ζ η θ ι κ λ μ ν ξ ο π ρ ς σ τ υ φ χ ω
 - Alfabeto cirílico: бвгжзийклмнпстуфхцчшщъыь
 - Alfabetos centro-europeos: Á â ă ä ĺ ć ç č é ę ë ě í î ď đ ń ň ó ô ŕ ř
 - ى و م ل ع ص س د خ ح ج ث ت ة ب ا ئ إ ؤ أ آ ء ك گ ژ چ :Alfabeto árabe ■
 - Alfabeto hebreo:תשרקצץפףעסנןמםלכךיטחזוהדגבא
 - Alfabetos asiáticos: 中文萬國碼際字出典フリ百科事典ィキペデア
 - Símbolos: £ Pts € № ¼ ½ ¾ 1/7 ← ↑ ↗ ⇒ ∀ ∂ ∃ ∄ 器 ⊠
 - Emoji: ② ② ② ⊕ ⊕ ۞ **② ≥ ♥ ♥ ∜ №** ⊕ **⑤ ≥ ↓ ♣ № ₩** ↓ **※**



Caracteres en C++

- Dos tipos
 - char
 - □ 1 *byte* (8 bits)
 - wchar_t
 - □ 2 bytes (16 bits) en GNU GCC



Caracteres

- □ char
- Dominio de valores
 - 95 caracteres
 - Letras del alfabeto inglés
 - Dígitos
 - □ Signos de puntuación
 - Otros símbolos
 - 33 caracteres de control

	0	@	Р	`	р
!	1	Α	Q	a	q
11	2	В	R	b	r
#	3	C	S	С	S
\$	4	D	Т	d	t
%	5	Ε	U	е	u
&	6	F	V	f	٧
I	7	G	W	g	W
(8	Ι	Χ	h	Х
)	9		Υ	i	У
*	•	J	Z	j	Z
+	,	K	[k	{
,	\	Ш	\		
-	=	М]	m	}
•	>	N	٨	n	2
/	?	0		0	



Caracteres

□ Representación externa en C++

```
'A'
             'b'
                    'B'
                           'z'
                 '3'
             '2'
                           '4'
                                  '5'
                                         '6'
'0'
      '9'
      '$'
             1%'
             '@'
                    1 / 1
```



Representación interna

- Codificación arbitraria en binario
 - Código ASCII
 - American Standard Code for Information Interchange
 - Estandarizada por la American Standards Association en 1963
- □ Ejemplo: 'A' se codifica con
 - la secuencia binaria 0100 0001
 - el código numérico 65



Representación interna

Código Carácter	Código Carácter	Código	Código Carácter	Código Carácter	Código Carácter	Código Carácter	Código Carácter
Código Carácte	Código Carácte	Código Carácte	Código Carácte	Código Carácte	Código Carácte	Código Carácte	Código Carácte
0 NUL	16 DLE	32	48 0	64 @	80 P	96 `	112 p
1 SOH	17 DC1	33 !	49 1	65 A	81 Q	97 a	113 q
2 STX	18 DC2	34 "	50 2	66 B	82 R	98 b	114 r
3 ETX	19 DC3	35 #	51 3	67 C	83 S	99 c	115 s
4 EOT	20 DC4	36 \$	52 4	68 D	84 T	100 d	116 t
5 ENQ	21 NAK	37 %	53 5	69 E	85 U	101 e	117 u
6 ACK	22 SYN	38 &	54 6	70 F	86 V	102 f	118 v
7 BEL	23 ETB	39 '	55 7	71 G	87 W	103 g	119 w
8 BS	24 CAN	40 (56 8	72 H	88 X	104 h	120 x
9 HT	25 EM	41)	57 9	73 I	89 Y	105 i	121 y
10 LF	26 SUB	42 *	58 :	74 J	90 Z	106 j	122 z
11 VT	27 ESC	43 +	59 ;	75 K	91 [107 k	123 {
12 FF	28 FS	44 ,	60 <	76 L	92 \	108 I	124
13 CR	29 GS	45 -	61 =	77 M	93]	109 m	125 }
14 SO	30 RS	46 .	62 >	78 N	94 ^	110 n	126 ~
15 SI	31 US	47 <i>/</i>	63 ?	79 O	95 _	111 o	127 DEL



Otras codificaciones de caracteres

- \square 8 bits \rightarrow 256 caracteres
 - Latin1 (ISO 8859-1), Latin0 (ISO 8859-15), Windows-1252
 - Página de códigos 850
- \square 16 bits \rightarrow 65 536 caracteres
 - UCS-2 (2-byte Universal Character Set)
- □ Longitud variable → ~137 000 caracteres definidos por Unicode
 - UTF-8
 - UTF-16



Universal Character Set (UCS)

- □ Estándar internacional ISO/IEC 10646 (~Unicode)
 - Define 136 000 caracteres abstractos
 - Cada carácter abstracto se identifica de forma precisa por un entero único: punto de código (code point)
 - Cada punto de código se puede codificar de acuerdo con distintas codificaciones:
 - □ UTF-8
 - 1, 2, 3 o 4 bytes
 - Compatible con los códigos ASCII de 7 bits
 - □ UTF-16
 - 2 o 4 bytes

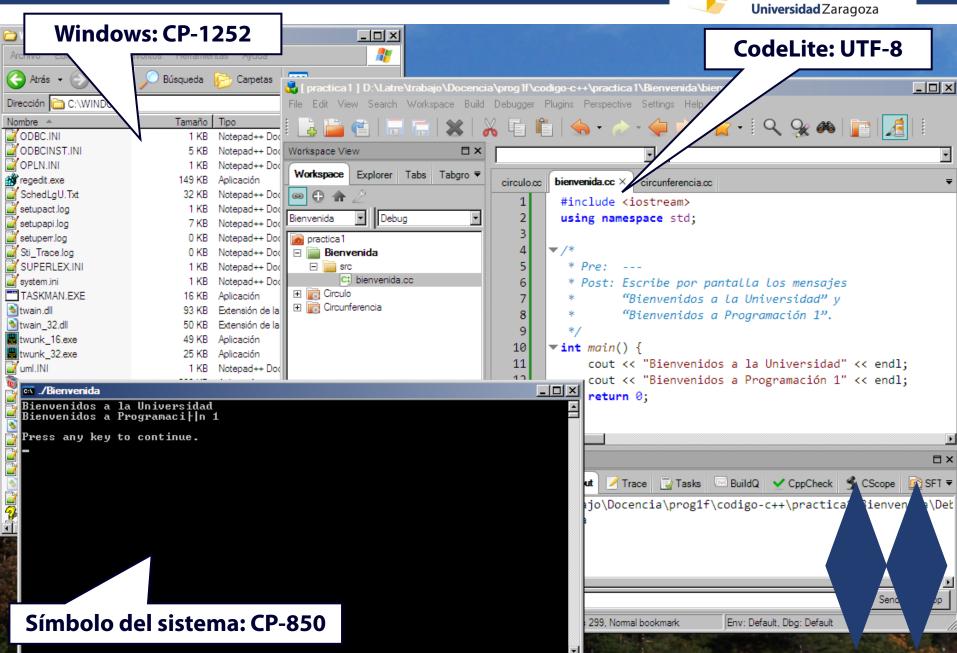




Problemas con las codificaciones

- □ Ejemplo 1:
 - Windows con CodeLite utilizando el símbolo del sistema de Windows
- □ Ejemplo 2:
 - Linux con CodeLite usando el terminal del SO
 - Windows con CodeLite utilizando el terminal de CodeLite







Ejemplo. Carácter ó

```
bienvenida.cc ×
                                              circulo.cc
                                                                  circunferencia.cc
                                                      #include <iostream>
                                                      using namespace std;
                                                4
                                                    ▼ /*
○ ✓ Bienvenida
                                                       * Pre: ---
Bienvenidos a la Universidad
                                                       * Post: Escribe por pantalla los mensajes
Bienvenidos a Programaci||r
                                                               "Bienvenidos a la Universidad" y
                                                               "Bienvenidos a Programación 1".
Press any key to continue.
                                               10
                                                    ▼int main() {
                                               11
                                                          cout << "Bienvenidos a la Universidad" << endl;</pre>
                                               12
                                                          cout << "Bienvenidos a Programación 1"
                                               13
                                                          return 0;
                                               14
```



Carácter «ó»

- □ Unicode:
 - «ó»
 - Descripción: Letra latina O minúscula con acento agudo
 - □ **Punto de código:** U+00F3 (en decimal: 243)
 - □ **Codificación en UTF-8:** bytes 195 y 179



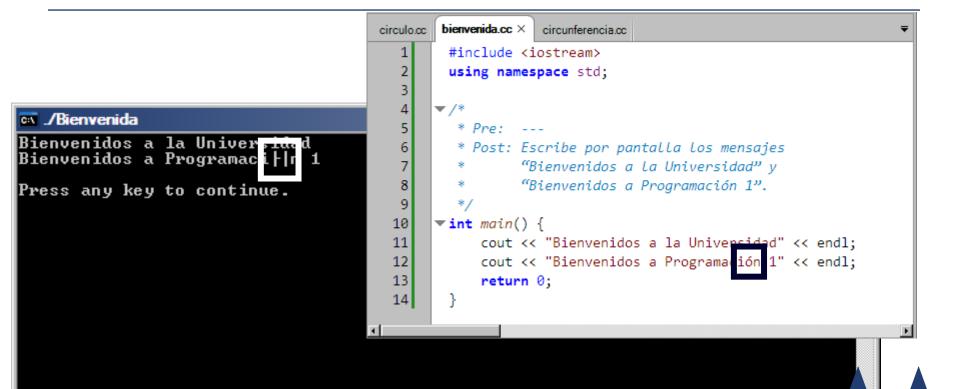


Página de códigos 850

128	Ç	129	ü	130	é	131	â	132	ä	133	à	134	å	135	Ç
136	ê	137	ë	138	è	139	ï	140	î	141	ì	142	Ä	143	Å
144	É	145	æ	146	Æ	147	ô	148	ö	149	ò	150	û	151	ù
152	ÿ	153	Ö	154	Ü	155	Ø	156	£	157	Ø	158	×	159	f
160	á	161	ĺ	162	ó	163	ú	164	ñ	165	Ñ	166	a	167	o
168	į	169	®	170	٦,	171	1/2	172	1/4	173	i	174	«	175	»
176	333 333	177	******	178		179		180	4	181	Á	182	Â	183	Á
184	©	185	4	186		187	7]	188]	189	¢	190	¥	191	٦
192	L	193	上	194	Т	195	F	196	_	197	+	198	ã	199	Ã
200	L	201	F	202	╨	203	ī	204	ŀ	205	=	206	#	207	¤
208	ð	209	Đ	210	Ê	211	Ë	212	È	213	ı	214	ĺ	215	Î
216	Ϊ	217	J	218	Γ	219		220		221	1	222	Ì	223	
224	Ó	225	ß	226	Ô	227	Ò	228	õ	229	Õ	230	μ	23	Þ
232	þ	233	Ú	234	Û	235	Ú	236	ý	237	Ý	238	-	2	
240		241	<u>±</u>	242	_	243	3/4	244	9	245	§	246	÷	24	3
248	0	249	••	250	•	251	1	252	3	253	2	254		255	18



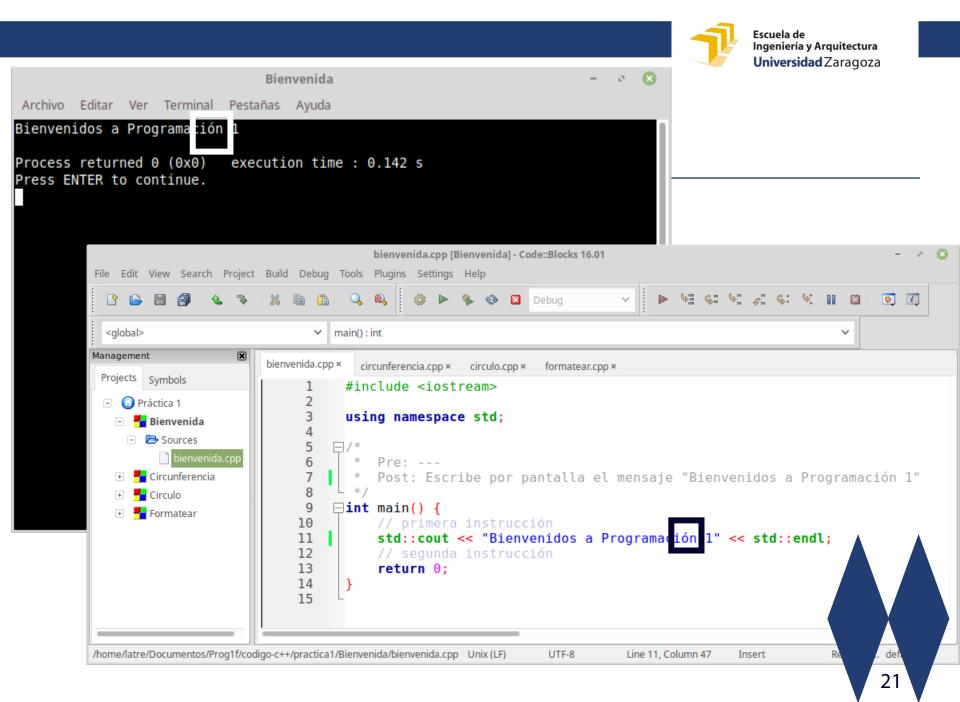
Carácter «ó»





Problemas con las codificaciones

- □ Ejemplo 1:
 - Windows con CodeLite utilizando el símbolo del sistema de Windows
- □ Ejemplo 2:
 - Linux con CodeLite usando el terminal del SO
 - Windows con CodeLite utilizando el terminal de CodeLite





Más información

- Joel Spolsky, «The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!)», Joel on Software, 8-10-2013.
 - https://www.joelonsoftware.com/2003/10/08/theabsolute-minimum-every-software-developerabsolutely-positively-must-know-about-unicode-andcharacter-sets-no-excuses/



Operadores asociados

- Los de los tipos enteros
 - Aritméticos: +, -, ...
 - Relación: ==, !=, <, <=, >, >=
- Conversión con enteros pueden ser explícitas:
 - int('A') devuelve 65
 - char(66) devuelve 'B'

Propiedades

- Hay secuencias de caracteres con códigos consecutivos crecientes:
 - Mayúsculas del alfabeto inglés: 'A', 'B', 'C', ..., 'X', 'Y' y 'Z'
 - Minúsculas del alfabeto inglés : 'a', 'b', 'c', ..., 'x', 'y' y 'z'
 - Dígitos: '0', '1', '2', '3', '4', '5', '6', '7', '8' y '9'

Expresiones con caracteres

```
□ char c = 'E';
 c == 'A'
□ c != 'e'
\Box c >= 'A'
□ c <= 'Z'
\Box c >= 'A' && c <= 'Z'
□ c >= 'a'
□ c <= 'z'
□ c >= 'a' && c <= 'z'
 char(c + 1)
   char(c + 32)
   char(c - 'A' + 'a')
```

```
Pre:
  Post: Si «c» es un carácter que
         representa una letra mayúscula
         entonces devuelve true; en otro
 *
         caso devuelve false.
*/
bool esMayuscula(const char c) {
    return c >= 'A' && c <= 'Z';
```

```
Pre:
  Post: Si «c» es un carácter que
         representa una letra minúscula
         entonces devuelve true; en otro
 *
         caso devuelve false.
*/
bool esMinuscula(const char c) {
    return c >= 'a' && c <= 'z';
```

```
Pre:
  Post: Si «c» es un carácter que
         representa un dígito entonces
         devuelve true; en otro caso
 *
         devuelve false.
 */
bool esDigito(const char c) {
    return c >= '0' && c <= '9';
```



Biblioteca estándar < cctype>

- Character handling functions. This header declares a set of functions to classify and transform individual characters.
 - isalnum: Check if character is alphanumeric
 - isalpha: Check if character is alphabetic
 - islower: Check if character is lowercase letter
 - isupper: Check if character is uppercase letter
 - isdigit: Check if character is decimal digit
 - isspace: Check if character is a white-space
 - tolower: Convert uppercase letter to lowercase
 - toupper: Convert lowercase letter to uppercase

```
* Pre:
  Post: Si «c» es un carácter que representa un dígito entonces
         devuelve el valor numérico comprendido entre 0 y 9
         representado por «c»;
         en otro caso devuelve un valor negativo.
 */
int valorDigito(const char c) {
    if (isdigit(c)) {
        return c - '0';
    else {
        return -1;
```



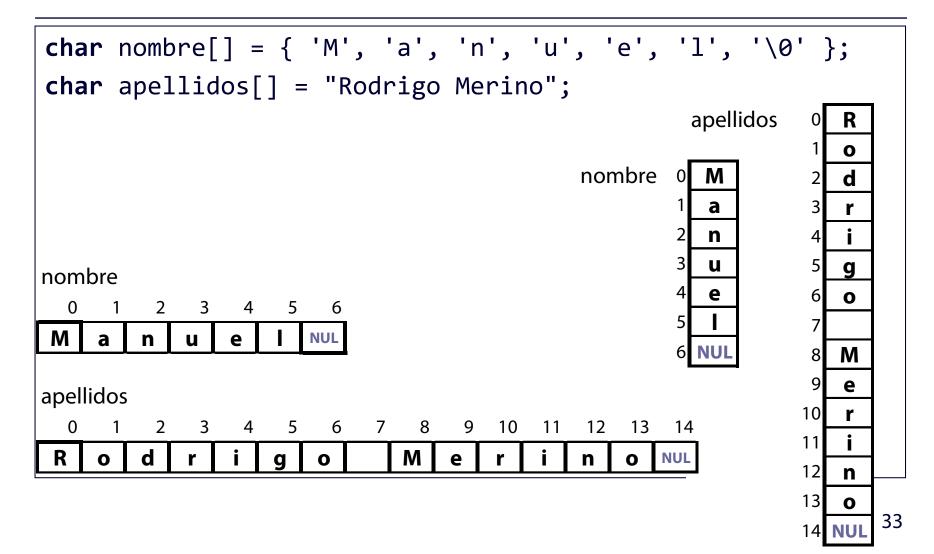
- □ Secuencia de 0, 1 o más caracteres
- Representación literal entre comillas

 - "A"
 - "Programación 1"



- Tipos de datos para su representación
 - Vector de datos de tipo char finalizadas con el carácter '\0'
 - □ Conocidas como *null-terminated strings* o *C strings*.
 - Clase predefinida string
 - No trabajaremos con ella en este curso
 - Más información:https://www.w3schools.com/cpp/cpp_strings.asp







```
#include <iostream>
using namespace std;
const int MAX LONG NOMBRE = 20;
  Programa que pide el nombre del usuario, lo lee del
 * teclado y lo escribe en la pantalla.
 */
int main() {
    cout << "Escriba su nombre: ";</pre>
    char nombre[MAX_LONG_NOMBRE];
    cin >> nombre;
    cout << "Su nombre es: " << nombre << endl;</pre>
    return 0;
```



```
#include <iostream>
using namespace std;
const int MAX_LONG_NOMBRE = 20;
   Programa que
 * teclado y lo
                    nombre
 */
int main() {
    cout << "Escriba su nombre: ";</pre>
    char nombre[MAX LONG NOMBRE];
    cin >> nombre;
    cout << "Su nombre es: " << nombre << endl;</pre>
    return 0;
```



```
#include <iostream>
using namespace std;
const int MAX_LONG_NOMBRE = 20;
  Programa que
 * teclado y lo
                    nombre
 */
int main() {
    cout << "Escriba su nombre;
    char nombre[MAX LONG NOMBRE];
    cin >> nombre; -
    cout << "Su nombre es: " << nombre << endl;</pre>
    return 0;
```

Cadenas de caracteres

- □ En la biblioteca predefinida <cstring>
 - strlen(cad)
 - devuelve la longitud de la cadena cad
 - strcpy(cad1, cad2)
 - □ copia la cadena cad2 en cad1
 - strcat(cad1, cad2)
 - modifica cad1 concatenándole la cadena cad2
 - strcmp(cad1, cad2)
 - compara las cadena cad1 y cad2
 - devuelve 0 si el contenido de ambas es idéntico
 - devuelve un valor positivo si cad1 es alfabéticamente posterior a cad2
 - devuelve un valor negativo si cad1 es alfabéticamente anterior a cad2

Ejemplo

```
char frase[80] = "";
cout << '\"' << frase << "\" tiene "
     << strlen(frase) << " caracteres." << endl;
char principio[] = "En un";
strcpy(frase, principio);
cout << '\"' << frase << "\" tiene "</pre>
     << strlen(frase) << " caracteres." << endl;
```

Ejemplo

```
char palabra[] = " lugar";
strcat(frase, palabra);
cout << '\"' << frase << "\" tiene "</pre>
     << strlen(frase) << " caracteres." << endl;
strcat(frase, " de la Mancha");
cout << '\"' << frase << "\" tiene "
     << strlen(frase) << " caracteres." << endl;
```



Ejemplo

```
"" tiene 0 caracteres.
"En un" tiene 5 caracteres.
"En un lugar" tiene 11 caracteres.
"En un lugar de la Mancha" tiene 24 caracteres.
```



Conversión de cadena a entero

```
«cadena» almacena una secuencia de caracteres
  Pre:
         que representan literalmente un entero con la
         siguiente sintaxis:
 *
            <literal entero> ::= [ <signo> ]
                                  <digito> { <digito> }
 *
            <signo> ::= "+" | "-"
 *
            <digito> ::= "0" | "1" | "2" | "3" |
 *
                       | "5" | "6" | "7" | "8" |
 *
  Post: Devuelve el valor entero representado en
         «cadena».
 */
int valorEntero(const char cadena[]);
```

Conversión de cadena a entero

```
int valorEntero(const char cadena[]) {
    int i = 0;
    int signo = 1;
    // cadena[0] es un dígito, signo '+' o signo
    if (cadena[0] == '+') {
        i++;
    else if (cadena[0] == '-') {
        i++;
        signo = -1;
```



Conversión de cadena a entero

```
int valorEntero(const char cadena[]) {
    // cadena[i] es un dígito
    int valor = 0;
    while (cadena[i] != '\0') {
        int valorDigito = cadena[i] - '0';
        valor = 10 * valor + valorDigito;
        i++;
    return signo * valor;
```



Conversión de entero a cadena

```
* Pre:
 * Post: Asigna a «literal» una secuencia de caracteres
         que representa literalmente el entero «valor»
         con la siguiente sintaxis:
            teral_entero> ::= [ <signo> ]
 *
 *
                                 <digito> { <digito> }
            <signo> ::= "+" | "-"
 *
            <digito> ::= "0" | "1" | "2" | "3"
 *
                       | "5" | "6" | "7" | "8" |
 *
 */
void literalEntero(const int valor, char literal[]);
```



Conversión de entero a cadena

```
void literalEntero(const int valor,
                   char literal[]) {
   // Cálculo del número de cifras.
   // Determinación de si es negativo o no.
   // Inserción del carácter de final de la
    // cadena.
   // Cálculo de las cifras de menos a más
          significativas, almacenándose en las
   //
   //
          componentes finales.
```



Conversión de entero a cadena

```
void literalEntero(const int valor, char literal[]) {
    // Cálculo del número de cifras
    int i = numCifras(valor);
    int n = valor;
    // Determinación de si valor es negativo o no
    if (valor < 0) {
        literal[0] = '-';
        n = -valor;
        i++;
    // Inserta el carácter de final de la cadena
    literal[i] = '\0';
```



Conversión de entero a cadena (otra versión)

```
void literalEntero(const int valor, char literal[]) {
    if (n == 0) {
        // Tratamiento específico del caso del 0;
        literal[0] = '0';
    else {
        // Cálculo de las cifras de «n» de menos a más
        // significativas
        while (n > 0) {
            i--;
            literal[i] = '0' + n % 10;
            n = n / 10;
```

Vectores de cadenas de caracteres

```
* Pre: 1 ≤ mes ≤ 12
 * Post: Ha escrito en la pantalla el nombre (en mayúsculas) del
         mes correspondiente al valor del parámetro «mes».
void escribirNombreMes(const int mes) {
    const int NUM MESES = 12;
    const int MAX LONG NOMBRE MES = 10;
    const char NOMBRES MES[NUM MESES][MAX LONG NOMBRE MES + 1]
            = { "ENERO", "FEBRERO", "MARZO", "ABRIL", "MAYO",
                "JUNIO", "JULIO", "AGOSTO", "SEPTIEMBRE",
                "OCTUBRE", "NOVIEMBRE", "DICIEMBRE" };
    cout << NOMBRES MES[mes - 1];</pre>
```