Arrays y Cadenas (String)

Sitio:	Centros - Cádiz	Imprimido por:	Barroso López, Carlos
Curso:	Programación	Día:	martes, 21 de mayo de 2024, 23:28
Libro:	Arrays y Cadenas (String)		

Tabla de contenidos

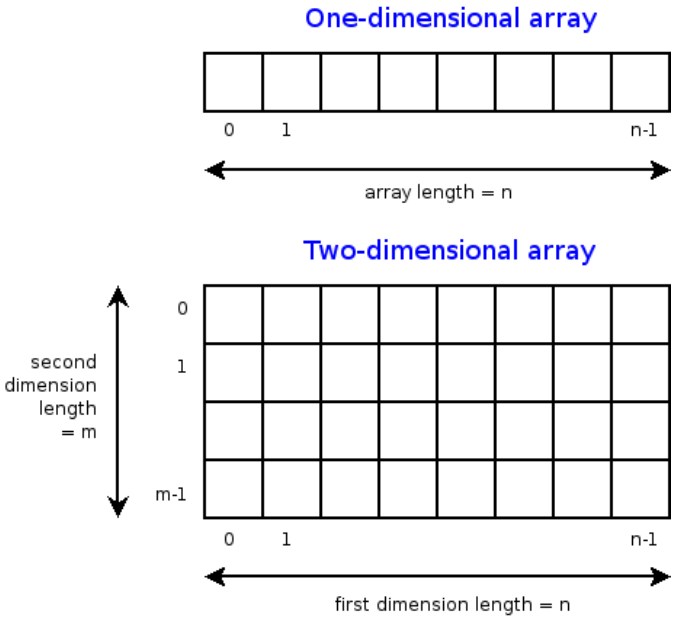
1. Arrays

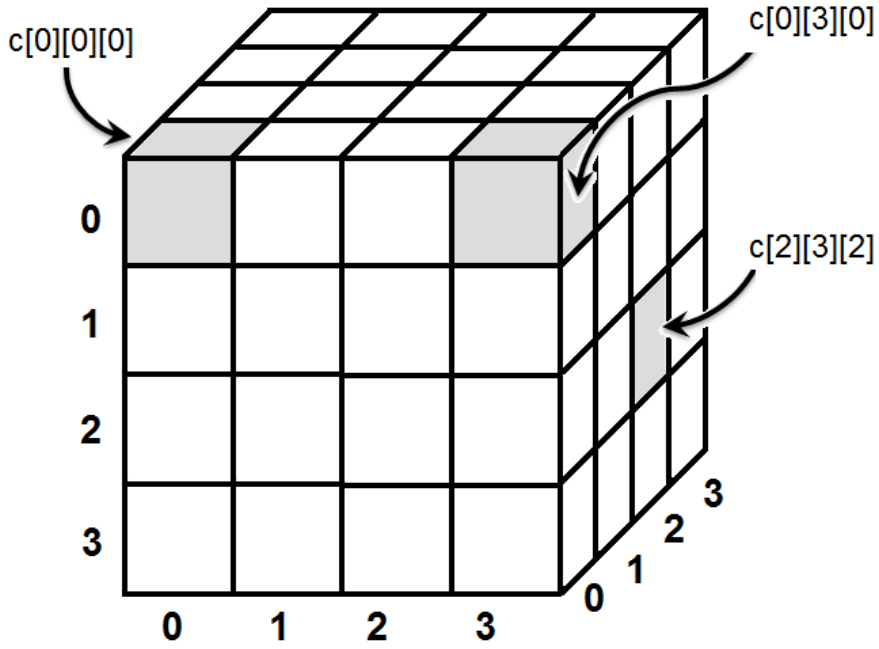
Un array es una estructura estática compuesta por un conjunto de datos del mismo tipo, que ocupan posiciones de memoria consecutivas.

En C++ se pueden declarar arrays de cualquier tipo básico del lenguaje: int, float, etc.

1. Características

- Los elementos de un array se referencian mediante su identificador y uno o varios índices.
- El número de índices determina las **dimensiones** del array: unidimensional o vector, bidimensional o matriz/tabla, y multidimensional (3 o más dimensiones)
- Un **índice** toma valores de **0 a N-1**, siendo N el tamaño del array en dicha dimensión.





2. Declaración

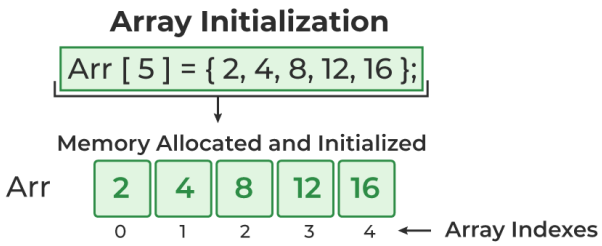
tipo identificador[tamaño1][tamaño2]...[tamañoN];

Ejemplos:

```
int a[10];  // array unidimensional o vector de 10 elementos (enteros)
float b[10][5];  // array bidimensional o matriz de 10x5 elementos (reales)
```

```
int c[100], d[27];  // declaración de dos arrays del mismo tipo en la misma línea
```

3. Inicialización



Dos formas:

```
int n[5] = {1, 2, 3, 4, 5};
o
int n[5] {1, 2, 3, 4, 5};  // recomendada
```

En el caso de dos dimensiones:

```
int m[2][2] {{1,2}, {3,4}};
```

Si no hay suficientes valores de inicialización, el resto de elementos se inicializan automáticamente a `0`.

```
int n[5]{1,2}  // equivale a {1, 2, 0, 0, 0}
```

```
int n[5]{0}  // inicializa a 0 todos los elementos
```

En el caso de dos dimensiones:

```
int m[2][2] {1,2};  // equivale a {{1,2}, {0,0}}
```

```
int m[2][2] {0};  // inicializa a 0 todos los elementos
```

Si el tamaño se omite, el compilador genera el tamaño de forma automática.

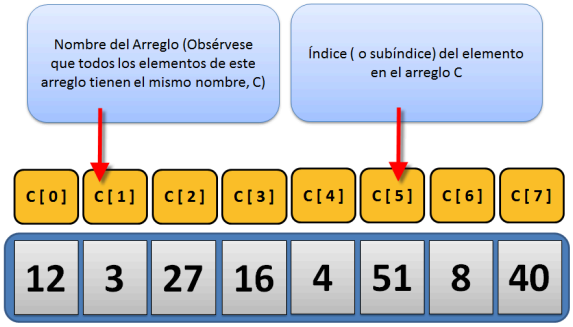
```
int n[]{1,2,3,4,5};  // equivale a int n[5]{1,2,3,4,5}
```

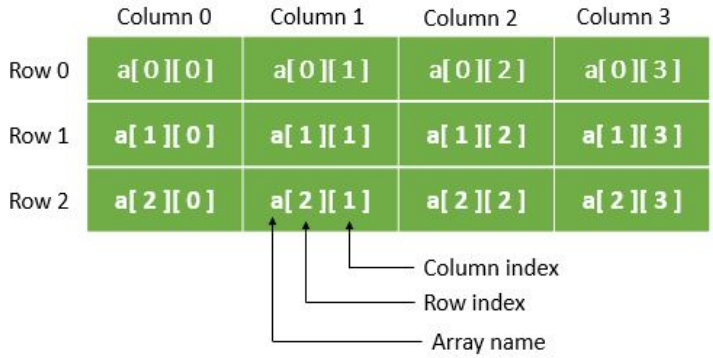
Si se añaden demasiados valores, el compilador generará un error de sintaxis.

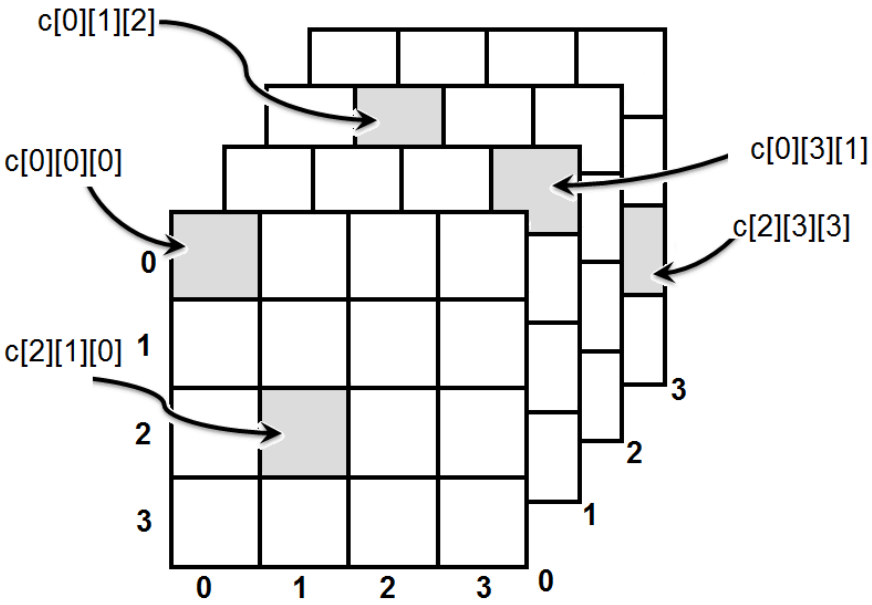
```
int n[5]{1,2,3,4,5,6};  // ¡ERROR!
```

4. Acceso a elementos

identificador[índice1][índice2]...[índiceN]







4.1. Ejemplo de acceso a los elementos de un vector

```
int v[5];  // array unidimensional v de 5 elementos
// v[0] primer elemento
// v[4] último elemento
v[0] = 3;  // se le asigna al primer elemento el valor 3
cout << v[0] << endl;  // los elementos de un array funcionan como cualquier otra variable
```

Pueden realizarse **operaciones dentro de los corchetes** siempre que el resultado de la expresión sea un valor entero:

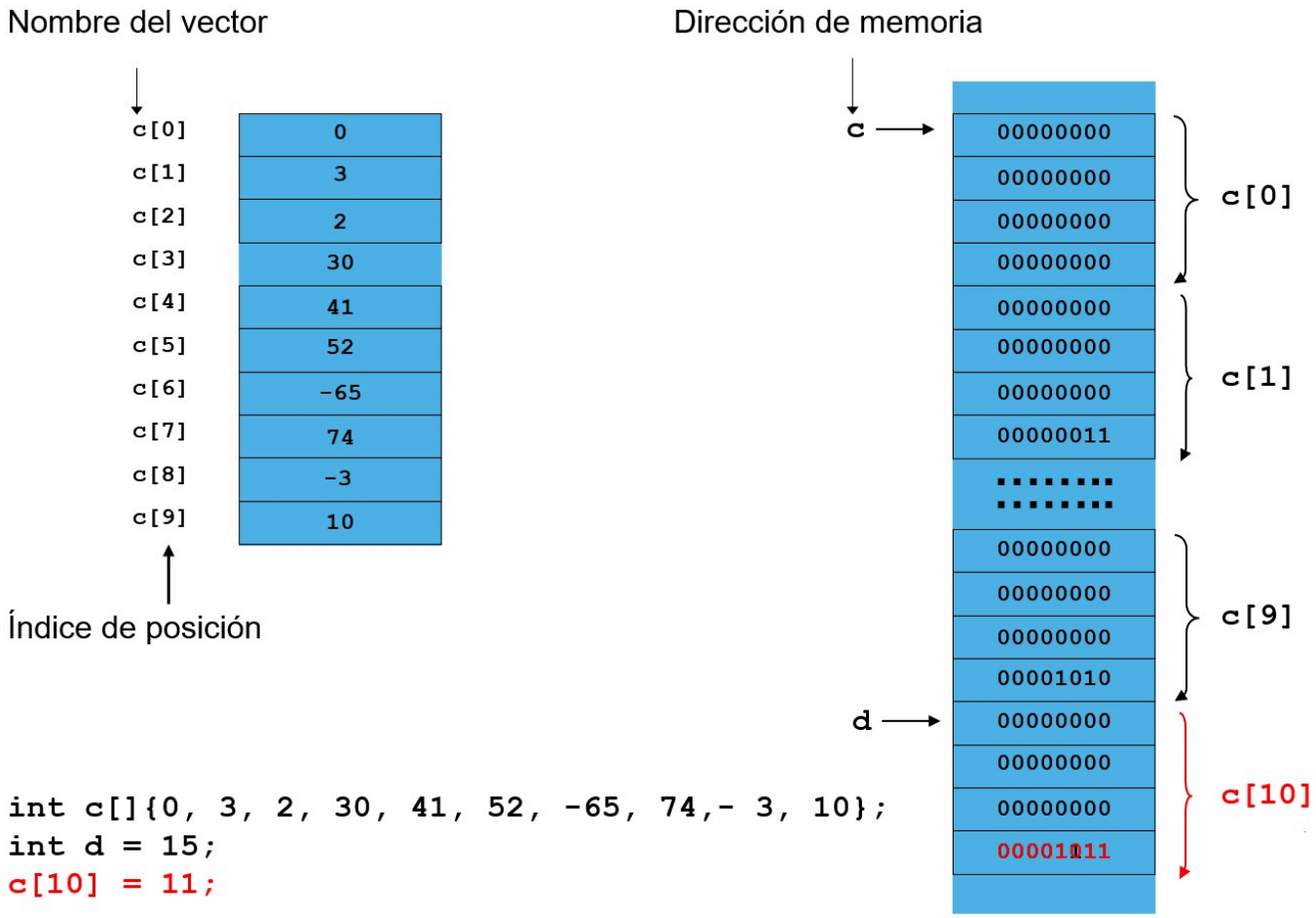
```
int i = 1;
v[i+1] = 5;  // equivale a v[2] = 5;
```

No es posible chequear si se sobrepasan los límites de un array. Por tanto, si se accede a una **posición no reservada**, puede estar leyéndose basura o escribiendo en posiciones de memoria incorrectas, provocando **efectos impredecibles** en la ejecución del programa.

```
cout << v[5] << endl;  // mostrará basura
```

```
v[5] = 3;  // asigna el valor 3 a una posición de memoria no reservada
```

4.2. Ejemplo de acceso a memoria no reservada



En este caso, se ha sobrescrito el valor de una variable. El programa tendrá un comportamiento imprevisible.

5. Recorrido

5.1. Recorrido de un vector

Vector de 10 elementos:

```
for(int i=0; i<10; i++)
  cout << v[i];
```

5.2. Recorrido de una tabla

Tabla de 10x10 elementos:

```
for(int i=0; i<10; i++) {
  for(int j=0; j<10; j++)
    cout << t[i][j];
}
```

i: recorre las filas

j: recorre las columnas

Ejercicios

Ejercicio 1: Pinta por pantalla el array v en forma de lista de elementos separados por comas.

Ejercicio 2: Pinta por pantalla la array t en forma de tabla (filas y columnas) separados por espacios.

Ejercicio 3: ¿Cómo recorrerías un array de 3 dimensiones? Pon un ejemplo.

6. Ventajas del uso de arrays

Ejemplo:

Leer por teclado 10 valores enteros y almacenarlos para su posterior procesamiento.

Versión sin arrays

```
int  a0, a1, a2, a3, a4, a5, a6, a7, a8, a9;
cout << "Introduzca dato 1 \n";
cin >> a0;
...
cout << "Introduzca dato 10 \n";
cin >> a9;
```

A pesar de ser una tarea repetitiva, no es posible usar bucles. Si el número de datos creciese, el programa se volvería ilegible e inmanejable.

Versión con arrays

```
int  a[10];
for (int i = 0; i < 10; i++)
{
   cout << "Introduzca el dato " << i+1 << ": ";
   cin >> a[i];
}
```

1.1. Ejemplo

Se solicita la edad de los alumnos de una clase y se calcula la media.

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

int pide_entero(int inf, int sup, string mensaje);

int main() // Calcula la edad media de una clase de alumnos
{
  const int num_max_alumnos = 100;
  int edad[num_max_alumnos];

  int num_alumnos = pide_entero(1, num_max_alumnos, "numero de alumnos");

  for(int i = 0; i < num_alumnos; ++i)
  {
    cout << "\nIntroduce la edad del alumno " << i+1 << ": ";
    cin >> edad[i];
  }

  double media = 0.;
  for(int i = 0; i < num_alumnos; ++i)
    media += edad[i];

  media /= num_alumnos;
  cout << "La edad media es " << media << endl;
}

int pide_entero(int inf, int sup, string mensaje)
{
  int valor;
  do
  {
    cout << "Introduzca el " << mensaje << ": ";
    cin >> valor;
    if (valor > sup)
      cout << "\n\nERROR: el " << mensaje << " no puede sobrepasar "
          << sup << "\n\n";
    else if (valor < inf)
      cout << "\n\nERROR: el " << mensaje << " debe ser al menos "
          << inf << "\n\n";
  }while (valor > sup || valor < inf);
  return valor;
}
```

La función `pide_entero()` es reutilizable para todos aquellos problemas que necesiten introducir un valor perteneciente a un intervalo.

1.2. Soluciones

Ejercicio 1

```
  int v[10]{1,2,3,-1};  // Inicialización de los primeros 4 elementos
  for(int i=0; i<10; i++) {
    cout << v[i];
    if (i<9)  // Añade una coma excepto en el último elemento
      cout << ",";
  }
```

Salida:

```
1,2,3,-1,0,0,0,0,0,0
```

Ejercicio 2

```
int t[5][10]{{1,2,3,4,5,6,7,8,9,10},{2,2,2,2,2,2,2,2,2,2}};
for(int i=0; i<5; i++) {
  for(int j=0; j<10; j++)
    cout << t[i][j] << " ";  // Pinta los elementos separados por un espacio

  cout << endl;  // Separa los elementos por filas
}
```

Salida:

```
1 2 3 4 5 6 7 8 9 10
2 2 2 2 2 2 2 2 2 2
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

Ejercicio 3

```
int m[3][5][5] {{{1,2,3,4,5},{6,7,8,9,10}},{{1,2,3,4,5},{6,7,8,9,10}}};
for (int i=0; i<3; i++) {
  for(int j=0; j<5; j++) {
    for (int k=0; k<5; k++)
      cout << m[i][j][k] << " ";
    cout <<endl;
  }
  cout <<endl;
}
```

Salida:

```
1 2 3 4 5
6 7 8 9 10
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0

1 2 3 4 5
6 7 8 9 10
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0

0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
```

2. Arrays en funciones

Una función puede utilizar como argumento un array, para ello:

1. Declaración

Prototipo:

```
tipo_funcion nombre_funcion(..., tipo_array [], ...);
```

Basta con indicar el tipo del array y los corchetes (tantos como dimensiones tenga).

2. Definición

```
tipo_funcion nombre_funcion(..., tipo_array nombre_array[], ...)
{
  ...
}
```

Debe indicarse además el identificador del array, como cualquier otro parámetro.

3. Llamada

```
{
  ...
  resultado = nombre_funcion(..., nombre_array, ...)
  ...
}
```

Basta con indicar el nombre del array.

4. Características

4.1. Identificador del array

Pasar el nombre del array en la llamada equivale a pasar la **dirección del primer elemento**. Por ello, la función sabe dónde está almacenado el array y puede modificar cualquiera de sus valores, como si de un "paso por referencia" se tratase.

Por tanto, no es necesario hacer una copia de todos los elementos del array, con el consiguiente ahorro en tiempo de ejecución y memoria, sobre todo para arrays de gran tamaño.

4.2. Tamaño del array

Dado que la función no posee información del tamaño del array recibido como parámetro, **el número de elementos suele ser un argumento** obligatorio más de la función.

2.1. Ejemplos

Ejemplo 1

Mínimo de un vector de enteros:

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

int minimo_vector(int [], int);
int main()
{
  int v[]{4,1,5,3};
  cout << minimo_vector(v, 4) << endl;
}
int minimo_vector(int v[], int num)
{
  int minimo = v[0];
  for (int i = 1; i < num; ++i)
    if (v[i] < minimo)
      minimo = v[i];
  return minimo;
}
```

Ejemplo 2

Mínimo y máximo de un vector de enteros:

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

void minimo_maximo_vector(int [], int, int&, int&);
int main()
{
  int v[]{4,1,5,3};
  int minimo, maximo;
  minimo_maximo_vector(v, 4, minimo, maximo);
  cout << "El mínimo y máximo del vector son "
      << minimo << " y " << maximo << endl;
}
void minimo_maximo_vector(int v[], int num, int& minimo, int& maximo)
{
  minimo = maximo = v[0];
  for (int i = 1; i < num; ++i)
    if (v[i] < minimo)
      minimo = v[i];
    else if (v[i] > maximo)
      maximo = v[i];
}
```

Ejemplo 3

Invertir el orden de los elementos de un vector:

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

void invertir_orden_vector(int v[], int num)
{
  for (int i = 0; i < num/2; ++i)  // Se recorre el vector hasta la mitad
  {
    int aux = v[i];
    v[i] = v[num-1-i];
    v[num-1-i] = aux;
  }
}
int pide_entero(int inf, int sup, string mensaje)
{
  int valor;
  do
  {
    cout << "Introduzca el " << mensaje << ": ";
    cin >> valor;
    if (valor > sup)
      cout << "\n\nERROR: el " << mensaje << " no puede sobrepasar " << sup << "\n\n";
    else if (valor < inf)
      cout << "\n\nERROR: el " << mensaje << " debe ser al menos " << inf << "\n\n";
  }while (valor > sup || valor < inf);
  return valor;
}
void carga_vector(int v[], int tam)
{
  for(int i = 0; i < tam; ++i)
  {
    cout << "v[" << i << "]= ";
    cin >> v[i];
  }
}
void muestra_vector(int v[], int tam)
{
  for (int i = 0; i < tam; ++i)
    cout << v[i] << " ";
  cout << endl;
}
int main()
{
  const int tam_max = 20;
  int v[tam_max];
  int tam = pide_entero(1, tam_max, "numero de elementos del vector");
  carga_vector(v, tam);
  invertir_orden_vector(v, tam);
  muestra_vector(v, tam);
}
```

3. Cadenas: La clase string

Una **cadena de caracteres** es un tipo especial de array unidimensional compuesto por elementos de tipo carácter.

C++ dispone de la clase `string` que permite trabajar con cadenas de caracteres.

1. Declaración e inicialización

Para declarar una variable de tipo cadena se utiliza el tipo de dato *string*, que en este caso es una clase.

Formas de declaración:

```
string cadena1;  // se inicializa por defecto a "" (cadena vacía)
string cadena2 = "Hola Mundo";  // declaración e inicialización
string cadena2 {"Hola Mundo"};  // forma equivalente
string cadena2 ("Hola Mundo");  // recomendada
```

1.1. Constructores

También es posible inicializar una cadena a partir de otra cadena, subcadena o carácter, utilizando diferentes tipos de constructores:

Constructor copia

```
string cadena1 (cadena2);  // se inicializa con el valor de cadena2
```

Constructor subcadena

```
string cadena1 (cadena2, 5);  // subcadena "Mundo" (desde la posición 5 hasta el final de cadena2)
string cadena1 (cadena2, 5, 3);  // subcadena "Mun" (desde la posición 5, 3 caracteres de cadena2)
```

Constructor buffer

```
string cadena1 ("Hola Mundo", 4);  // "Hola" (4 primeros caracteres del literal)
```

Constructor relleno

```
string cadena1 (6, 'a');  // "aaaaaa" (6 veces el carácter 'a')
```

Constructor rango

```
string cadena1 (cadena2.begin(), cadena2.begin()+4);  // "Hola"

(copia los caracteres de cadena2 en el rango [inicio, fin) )
```

Ejemplo:

```
// string constructor
#include <iostream>
#include <string>
using namespace std;
int main ()
{
  string s0 ("Initial string");
  // constructors used in the same order as described above:
  string s1;
  string s2 (s0);
  string s3 (s0, 8, 3);
  string s4 ("A character sequence");
  string s5 ("Another character sequence", 12);
  string s6a (10, 'x');
  string s6b (10, 42);      // 42 is the ASCII code for '*'
  string s7 (s0.begin(), s0.begin()+7);
  cout << "s1: " << s1 << "\ns2: " << s2 << "\ns3: " << s3;
  cout << "\ns4: " << s4 << "\ns5: " << s5 << "\ns6a: " << s6a;
  cout << "\ns6b: " << s6b << "\ns7: " << s7 << '\n';
  return 0;
}
```

Salida:

```
s1:
s2: Initial string
s3: str
s4: A character sequence
s5: Another char
s6a: xxxxxxxxxx
s6b: **********
s7: Initial
```

2. Acceso a elementos

El acceso a los elementos de una cadena (caracteres) se realiza mediante el **operador []**, como cualquier array:

```
cout << cadena2[0];  // acceso al primer carácter
H  // salida
```

También es posible acceder a los elementos mediante el **método at()**:

```
cout << cadena2.at(0);  // equivale a cadena2[0]
```

2.1. Acceso al primer y último elemento

Es posible acceder al primer y último elemento de la cadena mediante los métodos **front()** y **back()** respectivamente.

```
cout << cadena2.front();  // "H"
cout << cadena2.back();  // "o"
```

```
cadena2.front() = 'M';
cout << cadena2;  // "Mola Mundo"
```

3. Tamaño

Los métodos `length()` y `size()` devuelven el número de caracteres de una cadena.

Ejemplos:

```
cout << cadena2.length();
10
```

```
cout << cadena2[cadena2.length() - 1];  // acceso al último carácter
o
```

3.1. Cadena vacía

Es posible comprobar si una cadena está vacía mediante el método **empty()**:

```
if ( cadena2.empty() )  // equivale a cadena2.length() == 0
  cout << "Cadena vacía" << endl;
```

Y borrar todo el contenido de una cadena mediante el método **clear()**:

```
cadena2.clear();
cout << cadena2;  // No muestra nada, la cadena está vacía
```

4. Concatenación de cadenas

El operador + permite unir o concatenar dos o más cadenas de caracteres.

Ejemplo:

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

int main() {
        string nombre="Pepe";
        string apellidos="García";
        string nombre_completo;
        nombre_completo=nombre+" "+apellidos;
        cout << nombre_completo;
        return 0;
}
```

3.1. + Métodos (funciones miembro)

A continuación, se muestra el funcionamiento de los principales métodos o *funciones miembro* de la clase string:

```
string cadena="informática";
```

1. Modificadores

1.1. Añadir contenido

append(contenido_a_añadir)

Añade al final de la cadena el 'contenido' pasado por parámetro (ver constructores).

```
cadena.append(" moderna");  // Equivale a cadena += " moderna";
cout << cadena;
```

```
informática moderna
```

```
cadena.append(2, '!');  // añade 2 ocurrencias del carácter '!'
cout << cadena;
```

```
informática!!
```

push_back(carácter)

Añade al final de la cadena el 'carácter' pasado por parámetro.

```
cadena.push_back('!');
```

```
informática!
```

pop_back()

Elimina el último carácter de la cadena.

```
cadena.pop_back();
```

```
informátic
```

1.2. Asignar contenido

assign(contenido_a_asignar)

Asigna a la cadena el 'contenido' pasado por parámetro (ver constructores).

```
cadena.assign("Hola", 2);
```

```
Ho
```

1.3. Intercambiar contenido

swap(otra_cadena)

Intercambia los contenidos de la cadena y de la cadena pasada por parámetro.

```
string cadena2 ("Hola");
cadena.swap(cadena2);
cout << cadena << endl << cadena2 << endl;
```

```
Hola
Informática
```

1.4. Insertar, Borrar y Reemplazar

insert(posición, contenido_a_insertar)

Inserta en la 'posición' indicada el 'contenido' pasado por parámetro (ver constructores).

```
string sub="000";
cadena.insert(2, sub);
cout cadena;
```

```
in000formática
```

erase([posición], [cantidad])

Elimina de la cadena los caracteres desde la 'posición' indicada hasta el final, o la 'cantidad' indicada. Si no se indica la 'posición', borra todo el contenido (equivale a clear()).

```
cadena.erase(5, 2);  // borra la subcadena "má"
```

```
infortica
```

```
cadena.erase(5);  // borra la subcadena "mática"
```

```
infor
```

replace(posición, cantidad, contenido_a_reemplazar)

Reemplaza de la cadena la 'cantidad' de caracteres desde la 'posición' indicada por el contenido pasado por parámetro (ver constructores).

```
cadena.replace(6,5,"ación");
cout << cadena << endl;
```

```
información
```

2. Operaciones de cadena

2.1. Búsqueda

find(contenido_a_buscar, [posición])

Busca en la cadena la aparición del 'contenido a buscar', comenzando la búsqueda desde el principio de la cadena o desde la 'posición' indicada.

Devuelve:

Posición del primer carácter de la primera ocurrencia encontrada, o la constante **string::npos** si no encuentra ninguna ocurrencia.

El 'contenido a buscar' puede ser: una cadena, un literal o un carácter individual.

- Si el 'contenido a buscar' es un literal, puede añadirse un parámetro más con el número de caracteres del literal a considerar en la búsqueda.

find(literal_a_buscar, posición, num_caracteres)

```
cout << "Posición del primer carácter i: " << cadena.find("i") << endl;
cout << "Posición del segundo carácter i: " << cadena.find("i",1) << endl;
```

```
Posición del primer carácter i: 0
Posición del segundo carácter i: 8
```

rfind(contenido, [posición])

Similar a find() pero realizando la búsqueda de derecha a izquierda.

2.2. Comparación

compare([posición, tamaño], contenido_comparación)

Compara la cadena, o parte de ella (desde la 'posición' y 'tamaño' indicados), con un 'contenido de comparación' pasado por parámetro.

Devuelve:

```
0: si con iguales
-1: si la cadena es alfabéticamente MENOR que la cadena de comparación.
1: si la cadena es alfabéticamente MAYOR que la cadena de comparación.
```

```
string cadena{"Hola"};
cout << cadena.compare("Hola") << endl;  //  0
cout << cadena.compare("Holas") << endl; // -1 (después)
cout << cadena.compare("Hey") << endl;   //  1 (antes)
```

- Si el 'contenido de comparación' es otra cadena, pueden añadirse dos parámetros al final con la 'posición' y el 'tamaño' de la subcadena de comparación.

compare([posición, tamaño], cadena_comparación, [posición_cc, tamaño_cc])

- Si el 'contenido de comparación' es un literal, puede añadirse como parámetro el número de caracteres a considerar en la comparación.

compare([posición, tamaño], literal_comparación, [num_caracteres])

2.3. Subcadenas

substr([posición], [tamaño])

Devuelve la subcadena que comienza en la 'posición' y de 'tamaño' indicados por parámetro.

```
string subcad=cadena.substr(2,3);
cout << subcad << endl;
subcad=cadena.substr(5);
cout << subcad << endl;
```

```
for
mática
```

3.2. Lectura de cadenas: función getline()

1. Función getline()

La función getline() lee caracteres de un flujo de entrada y los almacena en una variable de tipo string, parando cuando encuentra el carácter delimitador (por defecto '\n' o salto de línea).

```
string linea;
getline(cin, linea);  // Lee de teclado una línea completa (hasta el salto de línea o tecla Enter)
```

Es posible cambiar el delimitador por defecto:

```
getline(cin, linea, ',');  // Lee de teclado hasta encontrar el carácter coma
```

2. Diferencias respecto al operador >>

La función getline() se utiliza para **leer líneas completas** de un flujo de entrada, tanto para leer de teclado como de un fichero.

Es importante recordar que el operador de extracción (>>) para cuando encuentra un delimitador cualquiera (como un espacio en blanco o un tabulador), y no necesariamente con el extremo de una línea de entrada.

Por tanto, la función getline() permite leer líneas del texto conteniendo espacios en blanco, tabuladores u otros delimitadores.

Ejemplo:

Leer por teclado "Hola Mundo" y almacenarlo en el string cadena.

Con operador >>

```
string cadena;
cin >> cadena;  // Se teclea "Hola Mundo" y se pulsa intro
cout << cadena;
```

```
Hola
```

Con getline()

```
getline (cin, cadena);  // Misma entrada
cout << cadena;
```

```
Hola Mundo
```

3. Vaciar buffer de entrada: cin.ignore()

En ocasiones no se lee todo el contenido de una entrada de datos, y puede que esos datos no leídos interfieran un las siguientes entradas de datos.

Ejemplo:

```
string cadena1, cadena2;
cin >> cadena1;  // Se teclea "Hola Mundo" y se pulsa intro
getline(cin, cadena2);  // ¡No pide nada por teclado! ¿Por qué?
cout << cadena1 << endl << cadena2 << endl;
```

```
Hola
 Mundo
```

Respuesta: La función getline() ha leído de los datos que aún quedaban en el buffer de entrada (" Mundo") y los ha almacenado en cadena2.

Para evitarlo, se puede vaciar el buffer de entrada mediante el método **cin.ignore()**:

```
string cadena1, cadena2;
cin >> cadena1;  // Se teclea "Hola Mundo" y se pulsa intro
cin.ignore();
getline(cin, cadena2);  // (Ahora sí pide una nueva entrada por teclado) Se teclea "Adiós"
cout << cadena1 << endl << cadena2 << endl;
```

```
Hola
Adiós
```

¿Cuándo es necesario?

Este comportamiento ocurre siempre que se lea de una entrada de datos mediante getline() y previamente se haya leído mediante el operador >> (incluso aunque se haya introducido una sola palabra), debido a que el operador >> deja en el buffer el salto de línea.

Ejemplo:

```
cin >> cadena1;  // Se teclea "Hola" y se pulsa intro
getline(cin, cadena2);  // No pide nada por teclado, para al leer el '\n' del buffer
cout << cadena1 << endl << cadena2 << endl;
```

```
Hola
[cadena 2 = "" (cadena vacía)]
```

También ocurre si se utiliza getline() con un delimitador diferente al salto de línea (por defecto), y posteriormente se utiliza getline() con el delimitador por defecto.

Ejercicio:

Crea un ejemplo y prueba este último caso.

¿Por qué no ocurre con el operador >>?

El operador de extracción >> **ignora los delimitadores** (saltos de línea, espacios, tabulaciones) que se encuentre **al inicio** de la entrada de datos.

3.3. Cadenas estilo C (C-string)

La cadena de caracteres de estilo C se originó en el lenguaje C y continúa siendo compatible con C++.

En C las cadenas de caracteres son en realidad un vector de elementos de tipo carácter. Utilizan el carácter especial `'\0'` para indicar el final de la cadena.

```
char cadena[] = {'H', 'o', 'l', 'a', '\0'};  // Declaración e inicialización
```

```
char cadena[]="Hola";  // Forma equivalente
```

```
cout cadena;
```

```
Hola
```

En ambos casos, cadena[4] = '\0'

3.4. Conversiones

1. Conversión de número a cadena

Función: string **to_string**(número)

```
cout << to_string(101) << endl;   // "101"
cout << to_string(3.14f) << endl; // "3.140000"
cout << to_string(45.32) << endl; // "45.320000"
```

2. Conversión de cadena a número

Funciones: int **stoi**(cadena), float **stof**(cadena), double **stod** (cadena)

```
cout << stoi("101") << endl;   // 101
cout << stof("3.14") << endl;  // 3.14
cout << stod("45.32") << endl; // 45.32
```

4. Ejemplo

Inicializar un vector de 5 cadenas a partir de los datos pedidos por teclado y posterior mostrarlos en pantalla y mostrar su longitud.

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

int main() {
    string vector[5];
    int i;

        for(i=0;i<5;i++)
        {
                cout << "Dime la cadena número "<< i+1<< ":";
                cin >> vector[i];
        }
        cout << "Las cadenas y sus longitudes" << endl;
        for(i=0;i<5;i++)
        {
                cout << vector[i] << ": " << vector[i].size() << endl;
        }
    return 0;
}
```

Reiniciar tour para usuario en esta página