Metodología de la Programación

Tema 1. Arrays, cadenas estilo C y matrices

Departamento de Ciencias de la Computación e I.A.

ETSIIT Universidad de Granada

Curso 2017-2018

Contenido del tema

Objetivos

Objetivos

- Conocer los mecanismos disponibles en C++ para almacener colecciones de valores de forma secuencial.
- Ser capaces de valorar el mecanismo más adecuado para un determinado problema.
- Conocer los conceptos básicos del uso de cadenas de caracteres estilo C (por compatibilidad entre C y C++).
- Comprender las matrices y saber realizar operaciones sobre ellas.

Importante: comprensión de los ejemplos de código y realización de los ejercicios que se vayan indicando.

Contenido del tema

Array

Un tipo de dato compuesto de un número fijo de elementos del mismo tipo y donde cada uno de ellos es directamente accesible mediante un índice.

`notas` =

`notas[0]`	`notas[1]`	...	`notas[499]`
2.4	4.9	...	6.7

Declaración de arrays

Declaración de un array

<tipo> <identificador> [<N.Componentes>];

Ejemplo: *double notas*[500];

Declaración de arrays

Declaración de un array

`<tipo> <identificador> [<N.Componentes>];`

Ejemplo: *double notas*[500];

- `<tipo>` indica el tipo de dato común a todas las componentes del array (double en el ejemplo).

Declaración de arrays

Declaración de un array

<tipo> <identificador> [<N.Componentes>];

Ejemplo: *double notas*[500];

- <tipo> indica el tipo de dato común a todas las componentes del array (double en el ejemplo).
- <identificador> nombre genérico para todas las componentes.

Declaración de arrays

Declaración de un array

`<tipo> <identificador> [<N.Componentes>];`

Ejemplo: *double notas*[500];

- `<tipo>` indica el tipo de dato común a todas las componentes del array (double en el ejemplo).
- `<identificador>` nombre genérico para todas las componentes.
- `<N.Componentes>` determina el número de componentes del array (500 en el ejemplo).

Declaración de arrays

Declaración de un array

<tipo> <identificador> [<N.Componentes>];

Ejemplo: *double notas*[500];

- <tipo> indica el tipo de dato común a todas las componentes del array (double en el ejemplo).
- <identificador> nombre genérico para todas las componentes.
- <N.Componentes> determina el número de componentes del array (500 en el ejemplo).
  - El número de componentes debe conocerse cuando se escribe el programa y no es posible alterarlo durante la ejecución del programa.

Declaración de arrays

Declaración de un array

<tipo> <identificador> [<N.Componentes>];

Ejemplo: *double notas*[500];

- <tipo> indica el tipo de dato común a todas las componentes del array (double en el ejemplo).
- <identificador> nombre genérico para todas las componentes.
- <N.Componentes> determina el número de componentes del array (500 en el ejemplo).
  - El número de componentes debe conocerse cuando se escribe el programa y no es posible alterarlo durante la ejecución del programa.
  - Pueden usarse literales o constantes enteras pero **nunca una variable**[1].

Declaración de arrays

Consejo

Usar constantes para especificar el tamaño de los arrays.
Ventajas:

- Es más fácil adaptar el código ante cambios de tamaño.
- Mejoramos la legibilidad y mantenibilidad del código.

```
const int NUM_ALUMNOS = 500;
double notas[NUM_ALUMNOS];
```

Declaración e inicialización de arrays

Declaración e inicialización de arrays

Podemos declarar e inicializar un array al mismo tiempo de la siguiente forma

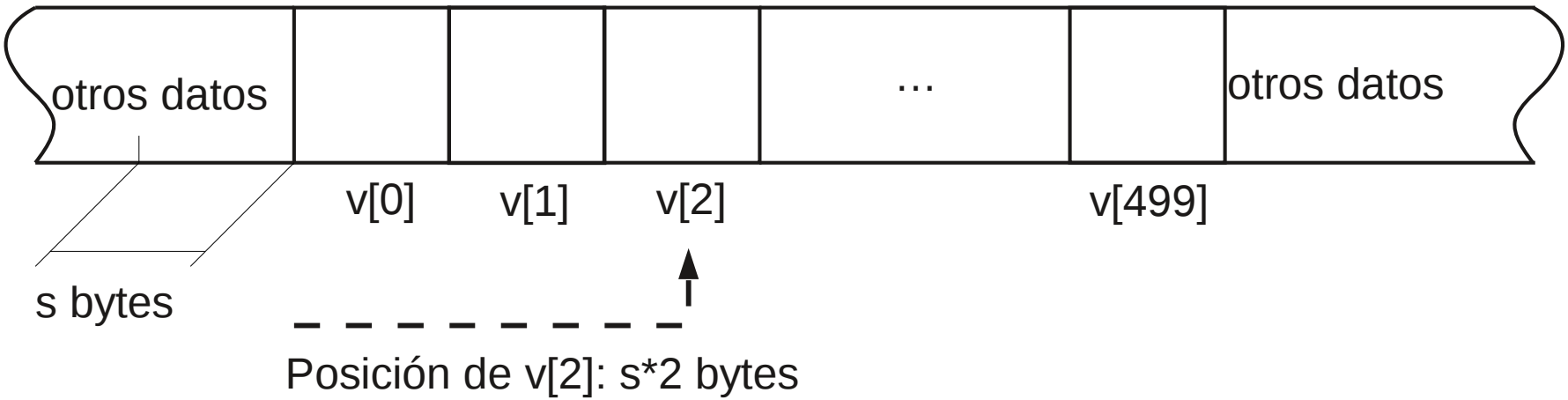
```
int array1[3] = {4,5,6};
int array2[7] = {3,5};
int array3[] = {1,3,9};
```

Almacenamiento en memoria de arrays

Almacenamiento en memoria de arrays

Las posiciones ocupadas por el array están contiguas en memoria.

```
double v[500];
```



Para acceder a la componente `i`, el compilador se debe desplazar `i` posiciones desde el comienzo del array.

Acceso a los elementos de un array

Acceso a los elementos de un array

Podemos acceder a cada elemento con la sintaxis:
<identificador> [<índice>]

- El índice de la primera componente del array es 0.

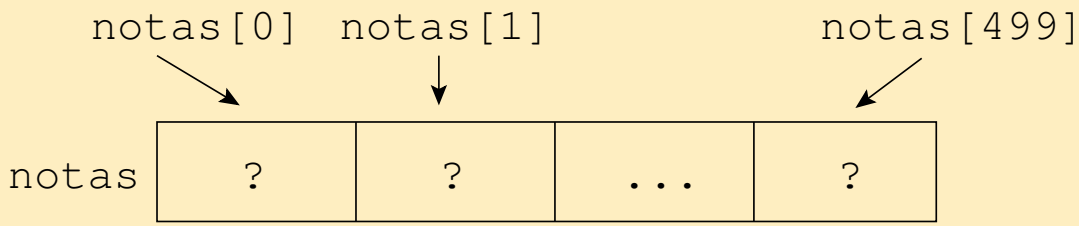
Acceso a los elementos de un array

Acceso a los elementos de un array

Podemos acceder a cada elemento con la sintaxis:
<identificador> [<índice>]

- El índice de la primera componente del array es 0.
- El índice de la última componente es `<N.Componentes>-1`.

```
double notas[500];
```



`notas[509]`, `notas['1']` o `notas[1.5]` **no son correctas**. El compilador no comprueba que los accesos sean correctos.

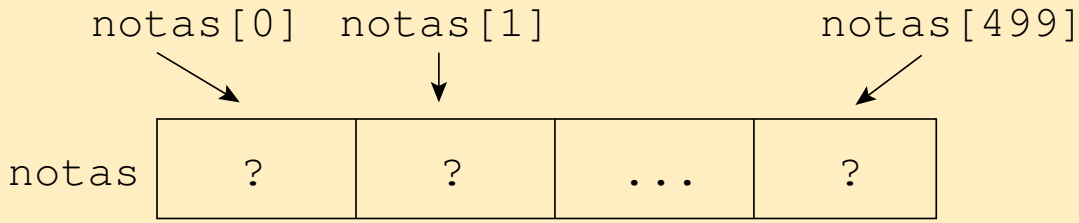
Acceso a los elementos de un array

Acceso a los elementos de un array

Podemos acceder a cada elemento con la sintaxis:
<identificador> [<índice>]

- El índice de la primera componente del array es 0.
- El índice de la última componente es `<N.Componentes>-1`.

```
double notas[500];
```



`notas[509]`, `notas['1']` o `notas[1.5]` **no son correctas**. El compilador no comprueba que los accesos sean correctos.

- Cada componente es una variable más del programa, del tipo indicado en la declaración del array.

Acceso a los elementos de un array

Ejemplo

```
const int NUM_ALUMNOS=500;
double notas[NUM_ALUMNOS];
...
for(int i=0;i<NUM_ALUMNOS;i++)
  cout<<notas[i]<<" ";
```

Asignación de valores a elementos del array

Asignación de valores

Debe hacerse elemento a elemento

```
notas[0]=5.7;
notas[1]=7.3;
```

Asignación completa

No está permitida la asignación completa

```
double notas[NUM_ALUMNOS];
double notas2[NUM_ALUMNOS];
...
notas2 = notas; // ERROR, esto no se puede hacer
```

Uso de una variable para controlar el número de elementos de un array

Control de elementos usados de un array

Habitualmente se usa una variable entera para controlar **el número de elementos usados** del array.

```
const int NUM_ALUMNOS = 500;
double notas[NUM_ALUMNOS];
int util_notas;
cout << "Introduce el numero de alumnos: ";
cin >> util_notas;
for(int i=0;i<util_notas;i++)
   cin >> notas[i];
```

Ejemplo de uso de arrays

Cálculo de nota media

Pediremos al usuario que indique el número de alumnos cuyas notas se van a procesar. Este valor se guarda en `util_notas`. Luego calculamos la nota media.

Subtareas a realizar:

- pregunta al usuario el número de alumnos a tratar
- bucle de lectura de notas
- bucle de calculo de media

```
1  int main(){
2      const int DIM_NOTAS = 100; // Maximo numero de notas
3      double notas[DIM_NOTAS];   // Array de almacenamiento
4      int util_notas; // Indica posiciones usadas del array
5      double media=0;
6      // Bucle de lectura de numero de alumnos: no puede
7      // ser negativo ni exceder la capacidad del array
8      do{
9        cout<<"Introduzca num. alumnos "
10           <<"(entre 1 y "<<DIM_NOTAS<<"): ";
11       cin >> util_notas;
12     }while (util_notas < 1 || util_notas > DIM_NOTAS);
13     // Bucle de lectura de las notas
14     for (int i=0; i<util_notas; i++){
15       cout << "nota[" << i << "]: ";
16       cin >> notas[i];
17     }
```

```
18      // Bucle acumulador
19      for (int i=0; i<util_notas; i++){
20        media += notas[i];
21      }
22      // Calculo de la media
23      media /= util_notas;
24      cout << "\nMedia: " << media << endl;
25  }
```

Control del tamaño de un array con un elemento centinela

Control del tamaño de un array con un elemento centinela

Otra forma de controlar el tamaño de los arrays (el número de elementos realmente almacenados en ellos) consiste en insertar un elemento *especial* (elemento centinela) al final del array.

Control del tamaño de un array con un elemento centinela

Control del tamaño de un array con un elemento centinela

Otra forma de controlar el tamaño de los arrays (el número de elementos realmente almacenados en ellos) consiste en insertar un elemento *especial* (elemento centinela) al final del array.

Debe tenerse en cuenta que:

Debe ser un valor que no sea posible (válido) dentro del conjunto de datos a almacenar.
Por ejemplo, para notas podríamos usar el valor -1 como marca de fin de almacenamiento de datos.

Ejemplo de uso de arrays con elemento centinela I

Ejercicio anterior de cálculo de nota media, mediante centinelas (-1, nota imposible ...).

```
1  int main(){
2      const int DIM_NOTAS = 100;
3      double notas[DIM_NOTAS];
4      double media;
5      int i;
6      cout << "nota[0]: (-1 para terminar): ";
7      cin >> notas[0];
8      for(i=1; notas[i-1] != -1 && i < DIM_NOTAS-1; i++){
9        cout << "nota[" << i << "]: (-1 para terminar): ";
10       cin >> notas[i];
11     }
```

Ejemplo de uso de arrays con elemento centinela II

```
12      if (i==DIM_NOTAS-1)
13        notas[i] = -1;
14
15      media=0;
16      for (i=0; notas[i] != -1; i++)
17         media += notas[i];
18
19      if (i == 0)
20         cout << "No se introdujo ninguna nota\n";
21      else{
22         media /= i;
23         cout << "\nMedia: " << media << endl;
24      }
25  }
```

Ejemplo de uso de arrays con elemento centinela III

Aspectos importantes del código anterior:

- ¿cuántos valores (notas) podemos realmente almacenar en el array?
- ¿es necesario asegurar el almacenamiento del valor -1 en la última posición?
- ¿por qué es necesario controlar que no se introdujo nota alguna?
- ¿habría algún error de compilación? ¿y de ejecución?
- ¿hemos elegido la estructura de control adecuada?

Contenido del tema

Peculiaridad de arrays miembros de una estructura

- No es válida la asignación directa de arrays:

```
int v1[50], v2[50];
v2 = v1;
```

  Se produce un error ya que las copias de arrays se deben hacer componente a componente.

- Sin embargo, sí que es válido lo siguiente:

```
struct vector50int{
    int util; // num de elementos usados
    int vector[50];
};
vector50int v1, v2;
...
v2 = v1;
```

Contenido del tema

Arrays y matrices de estructuras

El acceso a los diferentes miembros de las estructuras, se realiza combinando el acceso a los elementos de la matriz con las operaciones de acceso a los miembros de la estructura.

```
struct Alumno{
  string NIF;
  string nombre;
  int curso;
  char grupo;
  double notas[3];
};
int main(){
   Alumno listaAlumnos[100];
   listaAlumnos[0].NIF="26262727T";
   ...
   cin >> listaAlumnos[3].notas[0];
   ...
   listaAlumnos[1]=listaAlumnos[0];
```

Contenido del tema

Paso de argumentos: array

Las funciones son esenciales para descomponer un problema en subtareas, haciendo que cada función sea responsable de cierta parte del trabajo. Es fundamental conocer la forma en que se pasan arrays a funciones, tanto para entrada como para salida de datos.

Paso de argumentos: array

Las funciones son esenciales para descomponer un problema en subtareas, haciendo que cada función sea responsable de cierta parte del trabajo. Es fundamental conocer la forma en que se pasan arrays a funciones, tanto para entrada como para salida de datos.

El paso de arrays a funciones se hace mediante un parámetro formal que debe ser **exactamente** del mismo tipo (no basta con que sea compatible) que el parámetro actual.

Paso de argumentos: array

Función cuya responsabilidad será la de imprimir el contenido de un array de caracteres. El array de caracteres se pasa a la función como argumento.

```
1  #include <iostream>
2  using namespace std;
3
4  void imprime_array (char v[5]){
5     for (int i=0; i<5; i++)
6        cout << v[i] << " ";
7  }
8  int main(){
9     char vocales[5]={'a','e','i','o','u'};
10    imprime_array(vocales);
11 }
```

Paso de argumentos: array

Consideraciones:

- la función asume que el tamaño del array es 5. ¿Es general esto?
- ¿qué ocurre si deseamos imprimir un array de enteros? ¿sirve esta función? ¿se genera error de compilación?

Nota: si necesitamos usar el mismo método para diferentes tipos de datos, habrá que implementar una función para cada tipo.

Paso de argumentos: array

C++ permite usar un array sin dimensiones como parámetro formal.
Necesitamos saber el número de componentes usadas.

```
1 #include <iostream>
2 using namespace std;
3 void imprime_array(char v[], int util){
4    for (int i=0; i<util; i++)
5       cout << v[i] << " ";
6 }
7 int main(){
8    char vocales[5]={'a','e','i','o','u'};
9    char digitos[10]={'0','1','2','3','4',
10                     '5','6','7','8','9'};
11   imprime_array(vocales, 5); cout<<endl;
12   imprime_array(digitos, 10); cout<<endl;
13   imprime_array(digitos, 5); cout<<endl; // del '0' al '4'
14   imprime_array(vocales, 100); cout<<endl; // ERROR al ejecutar,
15 }                                          // no al compilar
```

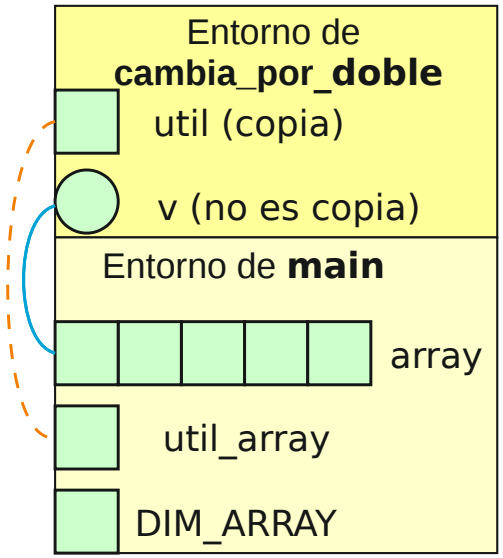
Paso de argumentos: array

El error de ejecución no se traduce siempre en un core... Puede que se muestren caracteres raros en pantalla (la conversión de las posiciones de memoria fuera del array a caracteres). Siempre hay que evitar esto, ya que el comportamiento del programa es impredecible.

```
a e i o u
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4
a e i o u   0 1 2 3 4 5 6 7 8 9 @   P %   8 C   m ( X %   s @
```

Los arrays se comportan como si se pasasen por referencia, en el sentido de que podemos modificar las componentes pero **no hay que poner** &.

```
1  #include <iostream>
2  using namespace std;
3
4  void imprime_array(int v[], int util){
5     for (int i=0; i<util; i++)
6        cout << v[i] << " ";
7  }
8  void cambia_por_doble(int v[], int util){
9     for (int i=0; i<util; i++)
10       v[i] *= 2;
11 }
12 int main(){
13    const int DIM_ARRAY = 5;
14    int array[DIM_ARRAY]={4,2,7};
15    int util_array=3;
16    cout << "Original: ";
17    imprime_array(array, util_array);
18    cout << endl << "Modificado: ";
19    cambia_por_doble(array, util_array);
20    imprime_array(array, util_array);
21 }
```



Paso de argumentos: array

La salida del programa anterior es la siguiente (observad que el array ha quedado modificado)

```
Original: 4 2 7
Modificado: 8 4 14
```

A considerar:

- ¿hay algún problema si el nombre del array, en el **main**, fuera arrayMain, por ejemplo?
- ¿hay algún problema si el argumento de las funciones llamado util se llamase también util_array?

Debe quedar clara la relación entre parámetros actuales y formales.

Paso de argumentos: array

Problema

¿Cómo evitamos que se puedan modificar los elementos contenidos en el array? ¿Interesa que el método que imprime el contenido del array pueda cambiar su contenido?

Paso de argumentos: array

Problema

¿Cómo evitamos que se puedan modificar los elementos contenidos en el array? ¿Interesa que el método que imprime el contenido del array pueda cambiar su contenido?

Solución: arrays de constantes

Utilizando el calificador `const`.

```
1 void imprime_array(const int v[], int util){
2     for (int i=0; i<util; i++)
3         cout << v[i] << " ";
4 }
5 void cambia_por_doble(const int v[], int util){
6     for (int i=0; i<util; i++)
7         v[i] *= 2; // ERROR de compilación
8 }
```

Paso de argumentos: array

Atención al error de compilación:

```
imprimedoble2.cpp: En la función 'void cambia_por_doble(const int*, int)':
imprimedoble2.cpp:10:15: error: asignación de la ubicación de sólo lectura
     '*(v + ((sizetype)(((long unsigned int)i) * 4ul)))'
```

Paso de argumentos: array

- Si no se utiliza el calificador `const`, el compilador asume que el array se va a modificar (aunque no se haga).
- No es posible pasar un array de constantes a una función cuya cabecera indica que el array se modifica (aunque la función realmente no modifique el array)

```
1  #include<iostream>
2  using namespace std;
3
4  void imprime_array (char v[]){
5     for (int i=0; i<5; i++)
6        cout << v[i] << " ";
7  }
8  int main(){
9     const char vocales[5]={'a','e','i','o','u'};
10    imprime_array(vocales); // ERROR de compilación
11 }
```

Paso de argumentos: array

Atención al error de compilación:

```
imprimevocalesconst.cpp: En la función 'int main()':
imprimevocalesconst.cpp:10:25: error: conversión inválida de
                                    'const char*' a 'char*' [-fpermissive]
imprimevocalesconst.cpp:4:6: error:   argumento de inicialización 1 de
                                  'void imprime_array(char*)' [-fpermissive]
```

Devolución de arrays por funciones I

Si queremos que una función devuelva un array, éste no puede ser local ya que al terminar la función, su zona de memoria *desaparecería*.
Debemos declarar dicho array en la función llamante y pasarlo como parámetro.

```
1  #include <iostream>
2  using namespace std;
3  void imprime_array(const int v[], int util);
4  void solo_pares(const int v[], int util_v,
5                  int pares[], int &util_pares);
6  int main(){
7     const int DIM=100;
8     int entrada[DIM] = {8,1,3,2,4,3,8}, salida[DIM];
9     int util_entrada = 7, util_salida;
10    solo_pares(entrada, util_entrada, salida, util_salida);
11    imprime_array(salida, util_salida);
```

Devolución de arrays por funciones II

```
12 }
13 void solo_pares(const int v[], int util_v,
14                  int pares[], int &util_pares){
15    util_pares=0;
16    for (int i=0; i<util_v; i++)
17       if (v[i]%2 == 0){
18          pares[util_pares]=v[i];
19          util_pares++;
20       }
21 }
22 void imprime_array(const int v[], int util){
23    for (int i=0; i<util; i++)
24       cout << v[i] << " ";
25 }
```

Ejemplo de devolución de un array por una función I

Ejemplo

Quitar los elementos consecutivos repetidos de un array, guardando el resultado en otro array.

```
1 #include <iostream>
2 using namespace std;
3
4 /**
5  * Metodo para imprimir vector: nos aseguramos que el vector
6  * no se modificara
7  * @param vector a imprimir
8  * @param numero de elementos en el vector
9  */
10
```

Ejemplo de devolución de un array por una función II

```
11 void imprime_array(const char v[], int util);
12 /**
13  * Metodo para quitar repetidos: solo si son valores consecutivos
14  * @param vector original
15  * @param contador de elementos en el vector original
16  * @param vector de destino
17  * @return contador de elementos en el vector resultado
18  */
19 int quita_repes(const char original[], int util_original, char destino[]);
20
21 // Metodo main
22 int main(){
23    const int DIM =100;
24    char entrada[DIM]={'b','b','i','e','n','n','n'}, salida[DIM];
25    int util_entrada = 7, util_salida;
```

Ejemplo de devolución de un array por una función III

```
26     // Se quitan los repetidos
27     util_salida=quita_repes(entrada, util_entrada, salida);
28     // Se muestra el vector
29     imprime_array(salida, util_salida);
30 }
31
32 // Quitar repetidos consecutivos
33 int quita_repes(const char original[], int util_original, char destino[]){
34   int util_destino=1;
35   // Se copia el primero tal cual
36   destino[0] = original[0];
37   // Bucle de recorrido del vector: desde la primera posicion
38   // en adelante. Se copia el valor si no es igual al previo
39   for (int i=1; i<util_original;i++){
40       if (original[i] != original[i-1]){
```

Ejemplo de devolución de un array por una función IV

```
41          destino[util_destino] = original[i];
42          util_destino++;
43       }
44    }
45    // Se devuelve el contador de elementos
46    return util_destino;
47 }
48
49 // Metodo de impresion
50 void imprime_array(const char v[], int util){
51     for (int i=0; i<util; i++){
52         cout << v[i] << " ";
53     }
54 }
```

Ejemplo de devolución de un array por una función V

```
Entrada: b b i e n n n
Salida: b i e n
```

Trabajando con arrays locales a funciones

Comprobar si un array de dígitos (0 a 9) de int es capicúa

Algoritmo:

1. Eliminar elementos que no estén entre 0 y 9.
2. Recorrer el array desde el principio hasta la mitad
   1. Comprobar que el elemento en la posición actual desde el inicio, es igual al elemento en la posición actual desde el final.

Trabajando con arrays locales a funciones

Comprobar si un array de dígitos (0 a 9) de int es capicúa

Algoritmo:

1. Eliminar elementos que no estén entre 0 y 9.
2. Recorrer el array desde el principio hasta la mitad
   1. Comprobar que el elemento en la posición actual desde el inicio, es igual al elemento en la posición actual desde el final.

Problema

Necesitamos un array local donde guardar el resultado del paso 1. ¿Cómo lo declaramos?

Trabajando con arrays locales a funciones

Comprobar si un array de dígitos (0 a 9) de int es capicúa

Algoritmo:

1. Eliminar elementos que no estén entre 0 y 9.
2. Recorrer el array desde el principio hasta la mitad
   1. Comprobar que el elemento en la posición actual desde el inicio, es igual al elemento en la posición actual desde el final.

Problema

Necesitamos un array local donde guardar el resultado del paso 1. ¿Cómo lo declaramos?
Lo ideal sería poder crear un array con el tamaño justo: el número de dígitos. Pero no sabemos cuántos habrá....

Trabajando con arrays locales a funciones

Así que habrá que usar una constante global

```
const int DIM = 100;

bool capicua(const int v[], int longitud){
    int solodigitos[DIM];
    ......
}
int main(){
    int entrada[DIM];
```

Trabajando con arrays locales a funciones

Así que habrá que usar una constante global

```
const int DIM = 100;

bool capicua(const int v[], int longitud){
    int solodigitos[DIM];
    ......
}
int main(){
    int entrada[DIM];
```

Es la única solución (de momento).

Trabajando con arrays locales a funciones

Así que habrá que usar una constante global

```
const int DIM = 100;

bool capicua(const int v[], int longitud){
    int solodigitos[DIM];
    ......
}
int main(){
    int entrada[DIM];
```

Es la única solución (de momento).
Inconveniente: no podemos separar la implementación de `capicua` de la definición de la constante.

Trabajando con arrays locales a funciones

Así que habrá que usar una constante global

```
const int DIM = 100;

bool capicua(const int v[], int longitud){
    int solodigitos[DIM];
    ......
}
int main(){
    int entrada[DIM];
```

Es la única solución (de momento).
Inconveniente: no podemos separar la implementación de `capicua` de la definición de la constante.
Solución: Memoria dinámica o clase `vector`.

```
1  #include <iostream>
2  using namespace std;
3
4  const int DIM = 100;
5
6  void quita_nodigitos(const int original[],
7       int util_original,int destino[], int &util_destino);
8  void imprimevector(const int v[], int util);
9  bool capicua(const int v[], int longitud);
10
11
12
13
14
15
16
17
```

```
18 int main(){
19     int entrada1[DIM]={1,2,3,4,3,2,1};
20     int util_entrada1=7;
21     int entrada2[DIM]={1,2,3,4,5,6,10,7,8,9,10,11,9,12,
22                        8,13,7,6,-1,5,4,3,2,1};
23     int util_entrada2=24;
24     imprimevector(entrada1, util_entrada1);
25     if (capicua(entrada1, util_entrada1))
26         cout << " es capicua\n";
27     else
28         cout << " no es capicua\n";
29     imprimevector(entrada2, util_entrada2);
30     if (capicua(entrada2, util_entrada2))
31         cout << " es capicua\n";
32     else
33         cout << " no es capicua\n";
34 }
```

```
35 void quita_nodigitos(const int original[],
36                       int util_original,
37                       int destino[], int &util_destino){
38    util_destino=0;
39    for (int i=0; i<util_original; i++)
40       if (original[i] > -1 && original[i] < 10){
41          destino[util_destino]=original[i];
42          util_destino++;
43       }
44 }
45
46 void imprimevector(const int v[], int util){
47   for (int i=0; i<util; i++)
48     cout << v[i] << " ";
49 }
50
51
```

```
52 bool capicua(const int v[], int longitud){
53    bool escapicua = true;
54    int solodigitos[DIM];
55    int long_real;
56
57    quita_nodigitos(v, longitud, solodigitos, long_real);
58
59    for (int i=0; i< long_real/2 && escapicua; i++)
60       if(solodigitos[i] != solodigitos[long_real-1-i])
61          escapicua = false;
62
63    return escapicua;
64 }
```

Contenido del tema

Cadenas de caracteres estilo C

Cadena de caracteres

Secuencia ordenada de caracteres de longitud variable.
Permiten trabajar con datos como apellidos, direcciones, etc...

Tipos de cadenas de caracteres en C++

1. **cstring**: cadena de caracteres heredado de C.
2. **string**: cadena de caracteres propia de C++ (estudiada en FP).

Cadenas de caracteres de C

Un array de tipo `char` de un tamaño determinado acabado en un carácter especial, el carácter `'\0'` (carácter nulo), que marca el fin de la cadena (véase ▸ uso del elemento centinela).

Literales de cadena de caracteres

Literal de cadena de caracteres

Es una secuencia de cero o más caracteres encerrados entre comillas dobles

Literales de cadena de caracteres

Literal de cadena de caracteres

Es una secuencia de cero o más caracteres encerrados entre comillas dobles

- Su longitud es el número de caracteres que tiene.
- Su tipo es un array de `char` con un tamaño igual a su longitud más uno (para el carácter nulo).

Literales de cadena de caracteres

Literal de cadena de caracteres

Es una secuencia de cero o más caracteres encerrados entre comillas dobles

- Su longitud es el número de caracteres que tiene.
- Su tipo es un array de `char` con un tamaño igual a su longitud más uno (para el carácter nulo).

`"Hola"` de tipo `const char[5]`

`"Hola mundo"` de tipo `const char[11]`

`""` de tipo `const char[1]`

Cadenas de caracteres: declaración e inicialización

```
char nombre[10] ={'J','a','v','i','e','r','\0'};
```

'J'	'a'	'v'	'i'	'e'	'r'	'\0'	?	?	?

```
char nombre[] ={'J','a','v','i','e','r','\0'}; // Asume char[7]
```

Equivalente a las anteriores son:

```
char nombre[10]="Javier";
char nombre[]="Javier";
```

Cadenas de caracteres: declaración e inicialización

```
char nombre[10] ={'J','a','v','i','e','r','\0'};
```

'J'	'a'	'v'	'i'	'e'	'r'	'\0'	?	?	?

```
char nombre[] ={'J','a','v','i','e','r','\0'}; // Asume char[7]
```

Equivalente a las anteriores son:

```
char nombre[10]="Javier";
char nombre[]="Javier";
```

¡Cuidado!

```
char cadena[]="Hola";                  //char[5]
char cadena[]={'H','o','l','a'}; // char[4]
```

Paso de cadenas a funciones

El paso de cadenas corresponde al paso de un array a una función.
Como la cadena termina con el carácter nulo, no es necesario especificar su tamaño.

Ejemplo

Función que nos diga la longitud de una cadena

```
1 int longitud(const char cadena[]){
2     int i=0;
3     while (cadena[i]!='\0')
4         i++;
5     return i;
6 }
```

Ejemplo

Función que concatena dos cadenas

```
1  void concatena(const char cad1[], const char cad2[],
2                 char res[]){
3     int pos=0;
4     for (int i=0;cad1[i]!='\0';i++){
5        res[pos]=cad1[i];
6        pos++;
7     }
8     for (int i=0;cad2[i]!='\0';i++){
9        res[pos]=cad2[i];
10       pos++;
11    }
12    res[pos]='\0';
13 }
```

Entrada/salida de cadenas

Para leer y escribir cadenas se pueden usar los operadores de lectura y escritura ya conocidos.

```
1 #include<iostream>
2 using namespace std;
3 int main(){
4     char nombre[80];
5     cout << "Introduce tu nombre: ";
6     cin >> nombre;
7     cout << "El nombre introducido es: " << nombre;
8 }
```

Entrada/salida de cadenas

Para leer y escribir cadenas se pueden usar los operadores de lectura y escritura ya conocidos.

```
1 #include<iostream>
2 using namespace std;
3 int main(){
4     char nombre[80];
5     cout << "Introduce tu nombre: ";
6     cin >> nombre;
7     cout << "El nombre introducido es: " << nombre;
8 }
```

Problema

`cin` salta separadores antes del dato y se detiene cuando encuentra un separador (saltos de línea, espacios en blanco y tabuladores). Es decir, no debe usarse para leer cadenas de caracteres que contengan espacios en blanco. Además, no consume el separador, que quedará pendiente para próximas operaciones de lectura.

Entrada/salida de cadenas

Solución: (si deseamos leer algún espacio en blanco)

`cin.getline(<cadena>, <tamaño>);`

Lee hasta que se encuentra un salto de línea o alcanza el límite de lectura.

Cuidado: al combinar el uso de cin y cin.getline hay que ser consciente dónde se dejará la lectura en cada momento.

Entrada/salida de cadenas

Solución: (si deseamos leer algún espacio en blanco)

`cin.getline(<cadena>, <tamaño>);`
Lee hasta que se encuentra un salto de línea o alcanza el límite de lectura.
Cuidado: al combinar el uso de cin y cin.getline hay que ser consciente dónde se dejará la lectura en cada momento.

```
1   char nombre[80],direccion[120];
2   int edad;
3   cout << "Introduce tu nombre: ";
4   cin.getline(nombre,80);
5   cout << "El nombre introducido es: " << nombre;
6   cout << "\nIntroduce tu edad: ";
7   cin >> edad;
8   cout << "La edad introducida es: " << edad;
9   cout << "\nIntroduce tu direccion: ";
10  cin.getline(direccion,120);
11  cout << "La direccion introducida es: " << direccion;
```

Entrada/salida de cadenas

```
Introduce tu nombre: Andrés Cano Utrera
El nombre introducido es: Andrés Cano Utrera
Introduce tu edad: 20
La edad introducida es: 20
Introduce tu direccion: La direccion introducida es:
```

Problema

`cin` se detiene cuando encuentra un separador, ¡y no lee el separador! (no lo consume y hace que `getline` dé por finalizada su operación al encontrarlo)

Entrada/salida de cadenas

Solución: Crear una función `lee_linea` que evite las líneas vacías

```
1 void lee_linea(char c[], int tamano){
2    do{
3       cin.getline(c, tamano);
4    } while (c[0] == '\0'); // equivale a } while(longitud(c)==0);
5 }
```

Entrada/salida de cadenas

Solución: Crear una función lee_linea que evite las líneas vacías

```
1 void lee_linea(char c[], int tamano){
2    do{
3       cin.getline(c, tamano);
4    } while (c[0] == '\0'); // equivale a } while(longitud(c)==0);
5 }
```

```
1    cout << "Introduce tu nombre: ";
2    lee_linea(nombre,80);
3    cout << "Introduce tu edad: ";
4    cin >> edad;
5    cout << "Introduce tu direccion: ";
6    lee_linea(direccion,120);
```

Conversión entre cadenas cstring y string

Podemos hacer fácilmente la conversión entre cadenas cstring y string

```
1  #include <iostream>
2  #include <string>
3  #include <cstring>
4  using namespace std;
5  int main(){
6      char cadena1[]="Hola";
7      string cadena2;
8      char cadena3[10];
9
10     cadena2=cadena1; // cstring-->string
11     strcpy (cadena3, cadena2.c_str()); // string-->cstring
12     cout<<"cadena2="<<cadena2<<endl;
13     cout<<"cadena3="<<cadena3<<endl;
14 }
```

La biblioteca cstring I

La biblioteca `cstring` proporciona funciones de manejo de cadenas de caracteres de C.
Entre otras:

- `char * strcpy(char cadena1[], const char cadena2[])`
  Copia `cadena2` en `cadena1`. Es el operador de asignación de cadenas.
- `int strlen(const char s[])`
  Devuelve la longitud de la cadena `s`.
- `char * strcat(char s1[], const char s2[])`
  Concatena la cadena `s2` al final de `s1` y el resultado se almacena en `s1`.

La biblioteca cstring II

- `int strcmp(const char s1[], const char s2[])`
  Compara las cadenas `s1` y `s2`. Si la cadena `s1` es menor (lexicográficamente) que `s2` devuelve un valor menor que cero, si son iguales devuelve 0 y en otro caso devuelve un valor mayor que cero.
- `conts char * strstr(const char s1[], const char s2[])`
  `char * strstr(char s1[], const char s2[])`
  Devuelve un puntero a la primera ocurrencia de `s2` en `s1`, o un puntero nulo si `s2` no es parte de `s1`.

La biblioteca cstring III

```
1 #include<iostream>
2 #include<cstring>
3 using namespace std;
4 int main(){
5     const int DIM=100;
6     char c1[DIM]="Hola";
7     char c2[DIM];
8
9     strcpy(c2, "mundo");
10    strcat(c1, " ");
11    strcat(c1, c2);
12
13    cout <<"Longitudes:"<<strlen(c1)<<" "<<strlen(c2);
14    cout << "\nc1: " << c1 << " c2: " << c2;
15
```

La biblioteca cstring IV

```
16   if (strcmp(c1,"adiós mundo cruel") < 0)
17     cout << "\nCuidado con las mayúsculas\n";
18
19   if (strcmp(c2, "mucho") > 0)
20     cout << "\n\"mundo\" es mayor que \"mucho\"\n";
21 }
```

Ejercicio: Problema 5.3 pág. 161 de A. Garrido

Implemente una función que reciba una cadena de caracteres, y la modifique para que contenga únicamente la primera palabra (considere que si tiene más de una palabra, están separadas por espacios o tabuladores).

```
1  #include <iostream>
2  #include <cstring>
3  using namespace std;
4
5  void lee_linea(char c[], int tamano);
6  void deja_solo_primera_palabra(char c[]);
7
8  int main() {
9      const int DIM=100;
10     char cadena[DIM];
11
12     cout << "Introduce una cadena: ";
```

```
13     lee_linea(cadena, DIM);
14     deja_solo_primera_palabra(cadena);
15     cout << "Resultado = " << cadena << endl;
16 }
17
18 void deja_solo_primera_palabra(char c[])  {
19   int i=0;
20   // No hay espacios en blanco al inicio
21   while (c[i] != ' ' && c[i] != '\t' && i < strlen(c))
22     i++;
23   if (i < strlen(c))
24     c[i] = '\0';
25 }
```

Ejercicio: Problema 5.6 pág. 161 de A. Garrido

Escriba una función que reciba una cadena de caracteres, una posición de inicio I y una longitud L, y que nos devuelva la subcadena que comienza en I y tiene tamaño L. Nota: Si la longitud es demasiado grande (se sale de la cadena original), se devolverá una cadena de menor tamaño.

```
1  #include <iostream>
2  #include <cstring>
3  using namespace std;
4
5  const int DIM=100;
6  void lee_linea(char c[], int tamano);
7  void recorta(const char c1[], int ini, int lon, char c2[]);
8
9  int main() {
10   char cadena1[DIM], cadena2[DIM];
11   int i, l;
```

```
12    cout << "Introduce una cadena: ";
13    lee_linea(cadena1, DIM);
14    cout << "Introduce el inicio y la longitud (enteros): ";
15    cin >> i >> l;
16    recorta(cadena1,i,l,cadena2);
17    cout << "Resultado = >" << cadena2 <<  endl;
18 }
19
20 void recorta(const char c1[], int ini, int lon, char c2[]){
21    int i=0;
22    while (i+ini<strlen(c1) //para que ini o lon no sean muy grandes
23           && i<lon)  {      // para contar hasta lon
24      c2[i] = c1[i+ini];
25      i++;
26    }
27    c2[i] = '\0';
28 }
```

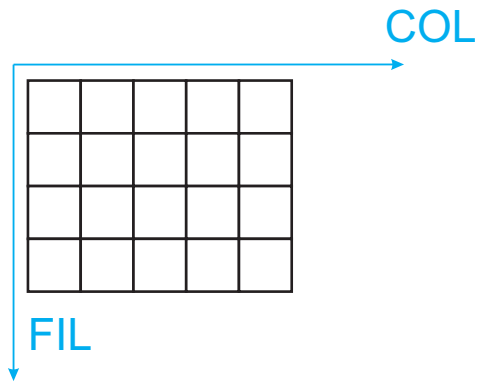
Contenido del tema

Declaración de matrices de 2 dimensiones

`<tipo> <identificador> [DIM_FIL][DIM_COL];`

- El tipo base de la matriz es el mismo para todas las componentes.
- Ambas dimensiones han de ser de tipo entero

```
1 int main(){
2     const int DIM_FIL = 2;
3     const int DIM_COL = 3;
4
5     double parcela[DIM_FIL][DIM_COL];
6 }
```



Inicialización

- "Forma segura": Poner entre llaves los valores de cada fila.

```
int m[2][3]={{1,2,3},{4,5,6}};      // m tendrá: 1  2  3
                                    //           4  5  6
```

Inicialización

- "Forma segura": Poner entre llaves los valores de cada fila.

```
int m[2][3]={{1,2,3},{4,5,6}};      // m tendrá: 1  2  3
                                    //           4  5  6
```

- Si no hay suficientes valores para una fila determinada, los elementos restantes se inicializan a 0.

```
int mat[2][2]={{1},{3,4}};      // mat tendrá: 1  0
                                //             3  4
```

Inicialización

- "Forma segura": Poner entre llaves los valores de cada fila.

```
int m[2][3]={{1,2,3},{4,5,6}};      // m tendrá: 1  2  3
                                    //           4  5  6
```

- Si no hay suficientes valores para una fila determinada, los elementos restantes se inicializan a 0.

```
int mat[2][2]={{1},{3,4}};      // mat tendrá: 1  0
                                //             3  4
```

- Si se eliminan los corchetes que encierran cada fila, se inicializan los elementos de la primera fila y después los de la segunda, y así sucesivamente.

```
int A[2][3]={1, 2, 3, 4, 5}  // A tendrá: 1  2  3
                             //           4  5  0
```

La declaración en detalle

- El compilador procesa las matrices como array de arrays.

La declaración en detalle

- El compilador procesa las matrices como array de arrays.
- Es decir, es un array con un tipo base también array (cada fila).

La declaración en detalle

- El compilador procesa las matrices como array de arrays.
- Es decir, es un array con un tipo base también array (cada fila).
- En la declaración
  `int m[2][3]`
  `m` es un array de 2 elementos (`m[2]`) y cada elemento es un array de 3 `int` (`int xxxx[3]`).

La declaración en detalle

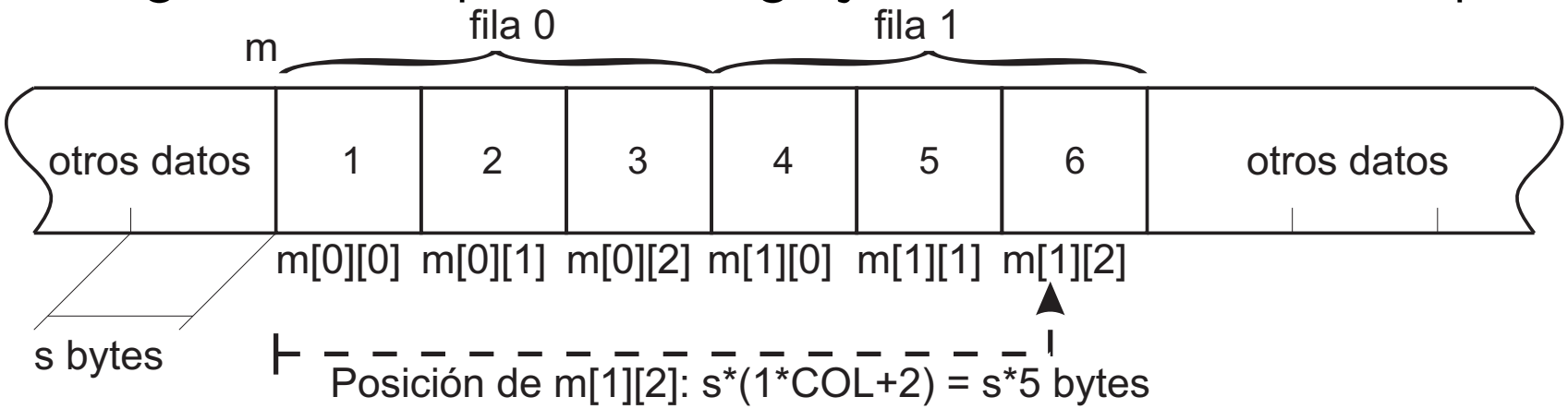
- El compilador procesa las matrices como array de arrays.
- Es decir, es un array con un tipo base también array (cada fila).
- En la declaración
  `int m[2][3]`
  `m` es un array de 2 elementos (`m[2]`) y cada elemento es un array de 3 `int` (`int xxxx[3]`).
- Observad que la sintaxis de la inicialización es la de un array de arrays
  `int m[2][3]={{1,2,3},{4,5,6}};`

Almacenamiento en memoria de matrices

Almacenamiento en memoria de los elementos de una matriz

Todos los elementos de las matrices se almacenan en un bloque contiguo de memoria.

- La organización depende del lenguaje: en C++ se almacenan por filas.

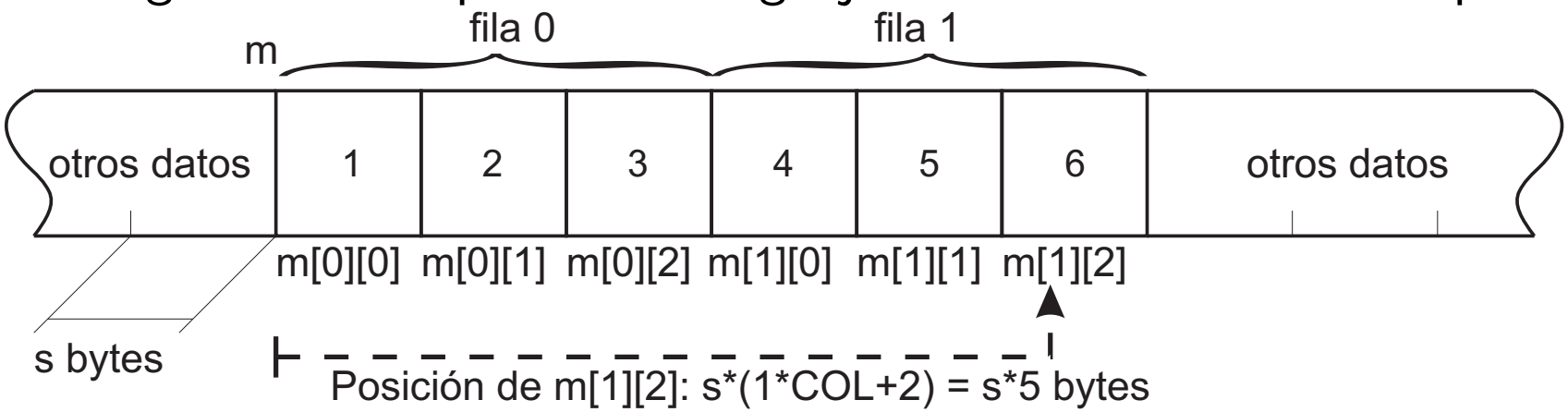


Almacenamiento en memoria de matrices

Almacenamiento en memoria de los elementos de una matriz

Todos los elementos de las matrices se almacenan en un bloque contiguo de memoria.

- La organización depende del lenguaje: en C++ se almacenan por filas.



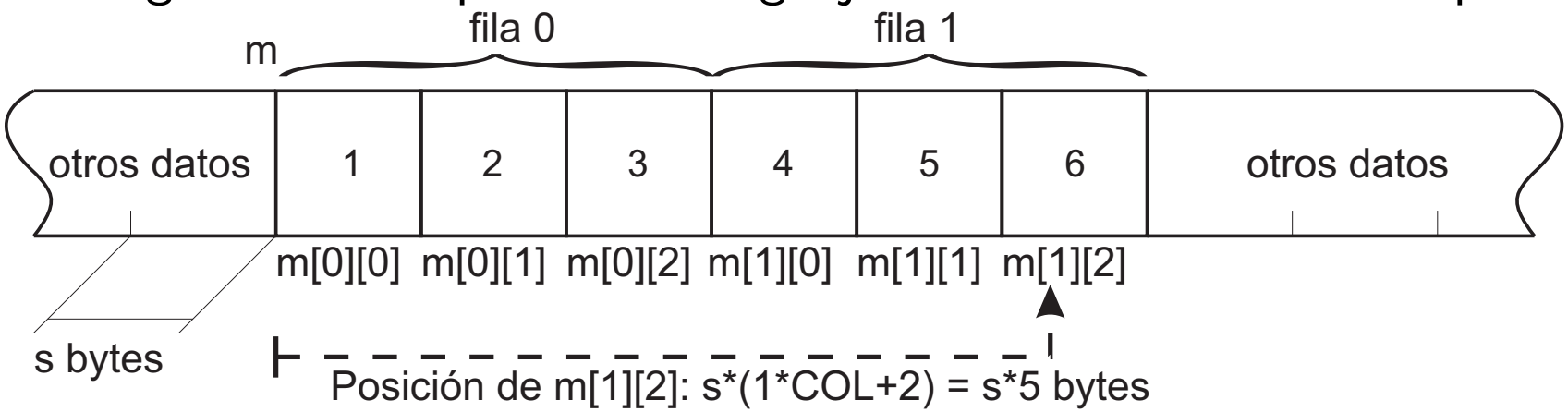
- Para acceder al elemento `m[i][j]` en una matriz `FIL` $\times$ `COL` el compilador debe *pasar a la fila* `i` y desde ahí moverse j elementos

Almacenamiento en memoria de matrices

Almacenamiento en memoria de los elementos de una matriz

Todos los elementos de las matrices se almacenan en un bloque contiguo de memoria.

- La organización depende del lenguaje: en C++ se almacenan por filas.



- Para acceder al elemento `m[i][j]` en una matriz `FIL` $\times$ `COL` el compilador debe *pasar a la fila* `i` y desde ahí moverse j elementos
- La posición del elemento `m[i][j]` se calcula como `i*COL + j`

Contenido del tema

Acceso, asignación, lectura y escritura

Acceso

<identificador> [<ind1>][<ind2>] (los índices comienzan en cero).
<identificador> [<ind1>][<ind2>] es una variable más del programa y se comporta como cualquier variable del tipo de dato base de la matriz.
¡El compilador no comprueba que los accesos sean correctos!

Acceso, asignación, lectura y escritura

Acceso

<identificador> [<ind1>][<ind2>] (los índices comienzan en cero).
<identificador> [<ind1>][<ind2>] es una variable más del programa y se comporta como cualquier variable del tipo de dato base de la matriz.
¡El compilador no comprueba que los accesos sean correctos!

Asignación

<identificador> [<ind1>][<ind2>] = <expresión>;
<expresión> ha de ser compatible con el tipo base de la matriz.

Acceso, asignación, lectura y escritura

Acceso

<identificador> [<ind1>][<ind2>] (los índices comienzan en cero).
<identificador> [<ind1>][<ind2>] es una variable más del programa y se comporta como cualquier variable del tipo de dato base de la matriz.
¡El compilador no comprueba que los accesos sean correctos!

Asignación

<identificador> [<ind1>][<ind2>] = <expresión>;
<expresión> ha de ser compatible con el tipo base de la matriz.

Lectura y escritura

cin >> <identificador> [<ind1>][<ind2>];
cout << <identificador> [<ind1>][<ind2>];

Contenido del tema

Sobre el tamaño de las matrices I

Para cada dimensión usaremos una variable que indique el número de componentes usadas.

```
1  #include <iostream>
2  using namespace std;
3  int main(){
4    const int FIL=20, COL=30;
5    double m[FIL][COL];
6    int fil_enc, col_enc, util_fil, util_col, f, c;
7    double buscado;
8    bool encontrado;
9    do{
10     cout << "Introducir el número de filas: ";
11     cin >> util_fil;
12   }while ((util_fil<1) || (util_fil>FIL));
```

Sobre el tamaño de las matrices II

```
13   do{
14     cout << "Introducir el número de columnas: ";
15     cin >> util_col;
16   }while ((util_col<1) || (util_col>COL));
17
18   for (f=0 ; f<util_fil; f++)
19     for (c=0 ; c<util_col ; c++){
20       cout << "Introducir el elemento ("
21            << f << "," << c << "): ";
22       cin >> m[f][c];
23     }
24   cout << "\nIntroduzca elemento a buscar: ";
25   cin >> buscado;
26   encontrado=false;
27
```

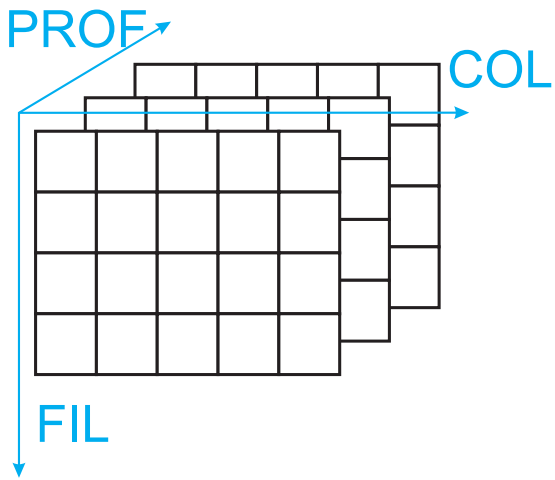
Sobre el tamaño de las matrices III

```
28    for (f=0; !encontrado && (f<util_fil) ; f++)
29      for (c=0; !encontrado && (c<util_col) ; c++)
30        if (m[f][c] == buscado){
31          encontrado = true;
32          fil_enc = f; col_enc = c;
33        }
34    if (encontrado)
35      cout << "Encontrado en la posición "
36        << fil_enc << "," << col_enc << endl;
37    else
38      cout << "Elemento no encontrado\n";
39    return 0;
40  }
```

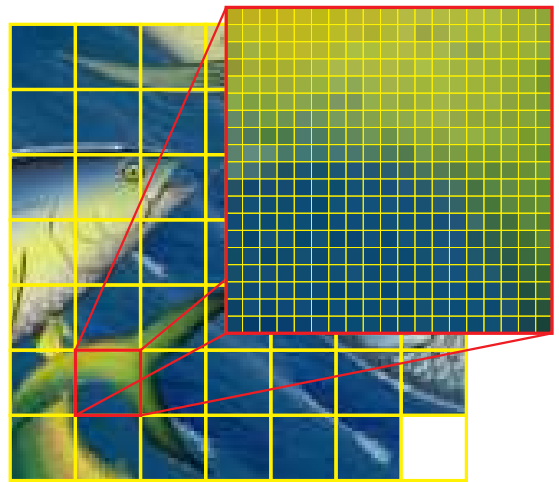
Contenido del tema

Matrices de más de 2 dimensiones

Podemos declarar tantas dimensiones como queramos añadiendo más corchetes.




```
1 int main(){
2     const int FIL = 4;
3     const int COL = 5;
4     const int PROF =3;
5     double mat[PROF][FIL][COL];
6
7     double puzle[7][7][19][19];
8 }
```



Contenido del tema

Funciones y matrices

Paso de matrices como parámetro de funciones y métodos

Para pasar una matriz hay que especificar todas las dimensiones menos la primera

- Ejemplo:

```
void lee_matriz(double m[][COL], int util_fil, int util_col);
```

Funciones y matrices

Paso de matrices como parámetro de funciones y métodos

Para pasar una matriz hay que especificar todas las dimensiones menos la primera

- Ejemplo:

```
void lee_matriz(double m[][COL], int util_fil, int util_col);
```

- COL no puede ser local a main. Debe ser global

```
const int FIL=20, COL=30;
void lee_matriz(double m[][COL], int util_fil, int util_col);

int main(){
  double m[FIL][COL];
  int util_fil=7, util_col=12;
  lee_matriz(m, util_fil, util_col);
```

Problema

Hacer un programa para buscar un elemento en una matriz 2D de doubles.

```
1  #include <iostream>
2  using namespace std;
3  const int FIL=20, COL=30;
4
5  void lee_matriz(double m[][COL],
6                  int util_fil, int util_col){
7    for (int f=0 ; f<util_fil; f++)
8      for (int c=0 ; c<util_col ; c++){
9        cout << "Introducir el elemento ("
10            << f << "," << c << "): ";
11       cin >> m[f][c];
12     }
13 }
14
15
```

```
16 void busca_matriz(const double m[][COL], int util_fil,
17                   int util_col, double elemento,
18                   int &fil_encontrado, int &col_encontrado){
19   bool encontrado=false;
20   fil_encontrado = -1; col_encontrado = -1;
21   for (int f=0; !encontrado && (f<util_fil) ; f++)
22     for (int c=0; !encontrado && (c<util_col) ; c++)
23       if (m[f][c] == elemento){
24         encontrado = true;
25         fil_encontrado = f;
26         col_encontrado = c;
27       }
28 }
29
30
31
32
```

```
33 int lee_int(const char mensaje[], int min, int max){
34   int aux;
35   do{
36     cout << mensaje;
37     cin >> aux;
38   }while ((aux<min) || (aux>max));
39   return aux;
40 }
41
42 int main(){
43   double m[FIL][COL];
44   int fil_enc, col_enc, util_fil,
45       util_col;
46   double buscado;
47
48   util_fil = lee_int("Introducir el número de filas: ",
49                      1, FIL);
```

```
50   util_col = lee_int("Introducir el número de columnas: ",
51                      1, COL);
52   lee_matriz(m, util_fil, util_col);
53   cout << "\nIntroduzca elemento a buscar: ";
54   cin >> buscado;
55
56   busca_matriz(m, util_fil, util_col, buscado,
57                fil_enc, col_enc);
58   if (fil_enc != -1)
59     cout << "Encontrado en la posición "
60       << fil_enc << "," << col_enc << endl;
61   else
62     cout << "Elemento no encontrado\n";
63
64   return 0;
65 }
```

Contenido del tema

Gestión de filas de una matriz como arrays I

Problema

Hacer una función que encuentre un elemento en una matriz 2D de doubles.

- Supongamos que disponemos de una función que permite buscar (búsqueda secuencial) un elemento en un array:

```
int busca_sec(double array[], int utilArray,
              double elemento);
```

- Dado que los elementos de cada fila están contiguos en memoria, podemos gestionar cada fila como si fuese un array y usar la función anterior para buscar.
- La fila `i`-ésima de una matriz `m` es `m[i]`.
- Cada fila `m[i]` tiene `util_col` componentes usadas

Gestión de filas de una matriz como arrays II

```
1  void busca_matriz(const double m[][COL], int util_fil,
2          int util_col, double elemento,
3          int &fil_enc, int &col_enc){
4    fil_enc = -1;
5    col_enc = -1;
6    for (int f=0; col_enc == -1 && (f<util_fil); f++)
7      col_enc = busca_sec(m[f], util_col, elemento);
8    if (col_enc !=-1)
9      fil_enc = f-1;
10 }
```

Gestión de filas de una matriz como arrays III

Otra solución

Como toda la matriz está contigua en memoria, si la matriz está completamente llena, podemos hacer

```
1  void busca_matriz(const double m[][COL], double elto,
2          int &fil_encontrado, int &col_encontrado){
3    int encontrado = busca_sec(m[0], COL*FIL, elto);
4    if (encontrado !=-1){
5      fil_encontrado = encontrado / COL;
6      col_encontrado = encontrado % COL;
7    else{
8      fil_encontrado = -1;
9      col_encontrado = -1;
10   }
11 }
```