Metodología de la Programación

Tema 1. Arrays, cadenas estilo C y matrices

Sylvia Acid (acid@decsai.ugr.es)
derivado de la obra de Andrés Cano
Departamento de Ciencias de la Computación e I.A.

Curso 2025-26

Índice I

1. Lo que sabemos hacer
2. Control de elementos usados de un array
   - Control del tamaño de un array con una variable
   - Control del tamaño de un array con un elemento centinela
3. Funciones y arrays
   - Paso de argumentos: array
   - Devolución de arrays por funciones
   - Trabajando con arrays locales a funciones
4. Cadenas de caracteres estilo C
5. Matrices de 2 dimensiones
   - Declaración e inicialización
   - Acceso, asignación, lectura y escritura
   - Sobre el tamaño de las matrices
   - Matrices de más de 2 dimensiones
6. Funciones y matrices
7. Gestión de filas de una matriz como arrays
8. Cuestiones abiertas

Contenido del tema

1. Lo que sabemos hacer
2. Control de elementos usados de un array
   - Control del tamaño de un array con una variable
   - Control del tamaño de un array con un elemento centinela
3. Funciones y arrays
   - Paso de argumentos: array
   - Devolución de arrays por funciones
   - Trabajando con arrays locales a funciones
4. Cadenas de caracteres estilo C
5. Matrices de 2 dimensiones
   - Declaración e inicialización
   - Acceso, asignación, lectura y escritura
   - Sobre el tamaño de las matrices
   - Matrices de más de 2 dimensiones
6. Funciones y matrices
7. Gestión de filas de una matriz como arrays
8. Cuestiones abiertas

Dadas las notas de los alumnos del aula

Se quiere:

1. hallar la nota máxima, y mínima
2. hallar la media
3. hallar la frecuencia de las calificaciones
4. hallar la moda de las calificaciones
5. hallar la varianza
6. se quieren las notas ordenadas

Qué estructura de datos es necesaria para cada problema?

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, y varianza

Subtareas a realizar:

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

```
1 int main(){
2    const int DIM_NOTAS = 100; // Maximo numero de notas a manejar
3    double notas[DIM_NOTAS];    // Array de almacenamiento de notas
4    int util_notas; // Indica posiciones usadas del array
5    double media=0;
6    double desviacion=0;
7    // Bucle de lectura de numero de alumnos: no puede ser negativo
8    // ni exceder la capacidad del array
9    do{
10     cout<<"Introduzca num. alumnos (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 de calculo de la media
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   // Bucle de calculo de la desvacion
26   for (int i=0; i<util_notas; i++){
27     varianza += (notas[i]-media)*(notas[i]-media);
28   }
29   // Calculo de la varianza
30   varianza /= util_notas;
31   cout << "\nMedia: " << media << endl;
32
33 }
```

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[99]`
2.4	4.9	...	6.7

Declaración de arrays

Declaración de un array

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

Ejemplo: *double notas*[100];

1

Declaración de arrays

Declaración de un array

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

Ejemplo: *double notas*[100];

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

1

Declaración de arrays

Declaración de un array

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

Ejemplo: *double notas*[100];

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

1

Declaración de arrays

Declaración de un array

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

Ejemplo: *double notas*[100];

- <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 (100 en el ejemplo).

1

Declaración de arrays

Declaración de un array

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

Ejemplo: *double notas*[100];

- <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 (100 en el ejemplo).
  - El número de componentes debe conocerse cuando se escribe el programa y no es posible alterarlo durante la ejecución del programa.

1

Declaración de arrays

Declaración de un array

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

Ejemplo: *double notas*[100];

- <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 (100 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].

[1] El estándar C99 permite usar una variable pero **C++ estándar no lo admite**. g++ lo admite como extensión propia.

Declaración de arrays

Importante

Usar constantes para especificar el tamaño de los arrays.
Ventaja: es más fácil adaptar el código ante cambios de tamaño.

```
const int NUM_ALUMNOS = 100;
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

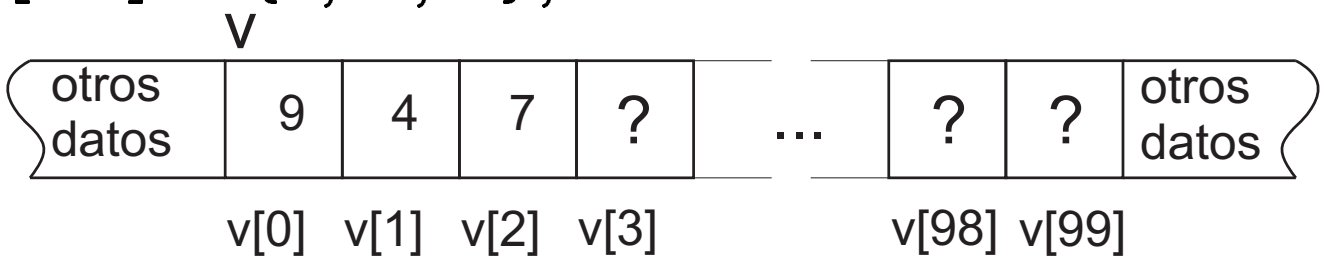
```
int array1[3] = {4,5,6};
int array2[7] = {3,5};
int array3[] = {1,3,9};
```

Almacenamiento en memoria de arrays

Almacenamiento en memoria de arrays

Las posiciones ocupadas por el array están contiguas en memoria.

```
double
    int v[100] = {9, 4, 7};
```



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

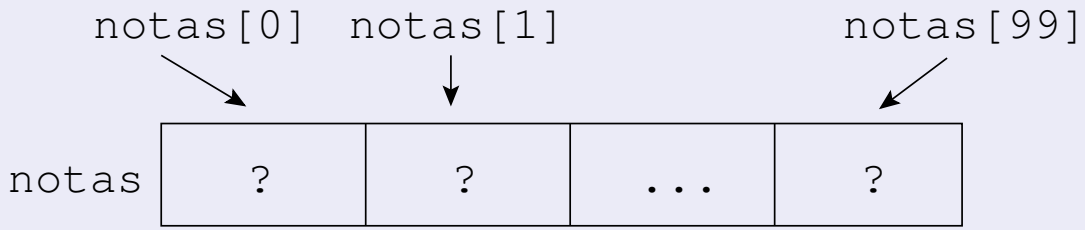
Acceso a los elementos de un array

Acceso a los elementos de un array

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

- Los valores válidos para índice son $[0, N.componentesReservados - 1]$

```
double notas[100];
```



`notas[-1]`, `notas['1']` o `notas[100]` **no son accesos válidos** pero, el compilador no comprueba rangos (no hay ningún aviso).
El acceso correcto es responsabilidad del programador.

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
```

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

Contenido del tema

1. Lo que sabemos hacer
2. Control de elementos usados de un array
   - Control del tamaño de un array con una variable
   - Control del tamaño de un array con un elemento centinela
3. Funciones y arrays
   - Paso de argumentos: array
   - Devolución de arrays por funciones
   - Trabajando con arrays locales a funciones
4. Cadenas de caracteres estilo C
5. Matrices de 2 dimensiones
   - Declaración e inicialización
   - Acceso, asignación, lectura y escritura
   - Sobre el tamaño de las matrices
   - Matrices de más de 2 dimensiones
6. Funciones y matrices
7. Gestión de filas de una matriz como arrays
8. Cuestiones abiertas

Control del tamaño de un array con una variable

Control del tamaño de un array

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

```
const int NUM_ALUMNOS = 100;
double notas[NUM_ALUMNOS];
int util_notas;
cout << "Introduce el numero de alumnos: ";
 ... // lectura de util_notas
     // lectura de notas
for(int i=0;i<util_notas;i++)
   cout << notas[i] << endl;
```

Valores válidos para el índice ya no es $[0, 99]$ sino $[0, util\_notas - 1]$.

Control de rango es responsabilidad del programador.

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

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
7     cout << "nota[0]: (-1 para terminar): ";
8     cin >> notas[0];
9     for(i=1; notas[i-1] != -1 && i < DIM_NOTAS-1; i++){
10      cout << "nota[" << i << "]: (-1 para terminar): ";
11      cin >> notas[i];
12    }
13    if (i==DIM_NOTAS-1)
14      notas[i] = -1;
15
16    media=0;
17    for (i=0; notas[i] != -1; i++)
18       media += notas[i];
19
20    if (i == 0)
21       cout << "No se introdujo ninguna nota\n";
22    else{
23       media /= i;
24       cout << "\nMedia: " << media << endl;
25    }
26 }
```

Ejemplo de uso de arrays con elemento centinela I

Aspectos importantes del código anterior:

- ¿cuántos valores (notas) podemos realmente almacenar en el array notas?
- ¿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?

Contenido del tema

1. Lo que sabemos hacer
2. Control de elementos usados de un array
   - Control del tamaño de un array con una variable
   - Control del tamaño de un array con un elemento centinela
3. Funciones y arrays
   - Paso de argumentos: array
   - Devolución de arrays por funciones
   - Trabajando con arrays locales a funciones
4. Cadenas de caracteres estilo C
5. Matrices de 2 dimensiones
   - Declaración e inicialización
   - Acceso, asignación, lectura y escritura
   - Sobre el tamaño de las matrices
   - Matrices de más de 2 dimensiones
6. Funciones y matrices
7. Gestión de filas de una matriz como arrays
8. Cuestiones abiertas

Paso de argumentos: array

Las funciones/métodos son esenciales para descomponer un problema en subtareas. El criterio: la cohesión. Haciendo que cada módulo sea responsable de **una tarea específica**.
Cómo se pasan los arrays como parámetros¿?
cuando son parámetros de entrada?? cuando son parámetros de salida¿?

Paso de argumentos: array

Las funciones/métodos son esenciales para descomponer un problema en subtareas. El criterio: la cohesión. Haciendo que cada módulo sea responsable de **una tarea específica**.
Cómo se pasan los arrays como parámetros¿?
cuando son parámetros de entrada?? cuando son parámetros de salida¿?

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

Paso de argumentos: array (sin centinela)

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

```
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 es 5. ¿Es una solución general?
- ¿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, aunque es necesario el control del tamaño del array.

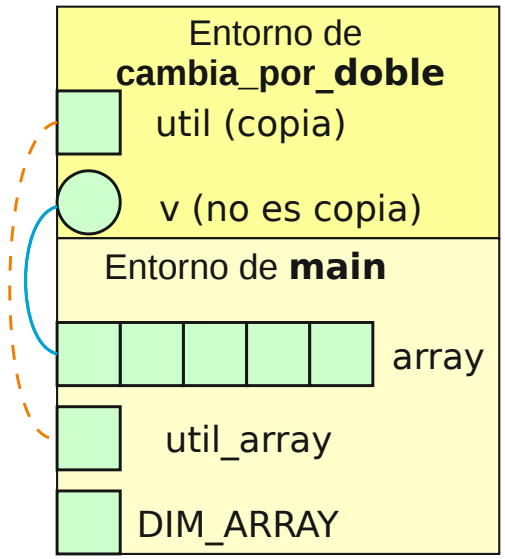
```
1  #include <iostream>
2  using namespace std;
3  void imprime_array(char v[], int util){
4     for (int i=0; i<util; i++)
5        cout << v[i] << " ";
6  }
7  int main(){
8     char vocales[5]={'a','e','i','o','u'};
9     char digitos[10]={'0','1','2','3','4', '5','6','7','8','9'};
10    imprime_array(vocales, 5); cout<<endl;
11    imprime_array(digitos, 10); cout<<endl;
12    imprime_array(digitos, 5); cout<<endl; // del '0' al '4'
13    imprime_array(vocales, 100); cout<<endl; // !!! ERROR ejecucion,
14
15 }
```

Paso de argumentos: array

El error en 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). El control de rangos válidos es indispensable, 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 @
```

```
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

modificado)

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

Los arrays pasan por referencia

Por defecto, los arrays pasan por referencia, sin introducir & en la cabecera. Lo que implica que se pueden modificar las componentes¡¡!!!

- Funciones que modifican arrays reciben éstos como parámetros de entrada/salida.
- Funciones que solo consultan arrays, debe recibir éstos protegidos contra escritura.

Paso de argumentos: array como parámetro de entrada

const

Utilizando el calificador const ante el parámetro para prevenir la escritura.

```
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 como parámetro de entrada

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 **es modificable** (aunque no se haga ninguna modificación).
- No es posible pasar un array de constantes a una función cuya cabecera indica que el array es **modificable**, esto es, indica que tiene permiso para escribir.

```
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

Para que una función devuelva un array (array de salida), éste no puede ser local ya que, al terminar la función, *su espacio desaparecería*. Los arrays de salida, son en realidad de entrada/salida.

```
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);
12 }
```

Devolución de arrays por funciones II

```
1  void solo_pares(const int v[], int util_v,
2                  int pares[], int &util_pares){
3     util_pares=0;
4     for (int i=0; i<util_v; i++)
5        if (v[i]%2 == 0){
6           pares[util_pares]=v[i];
7           util_pares++;
8        }
9  }
10 void imprime_array(const int v[], int util){
11    for (int i=0; i<util; i++)
12       cout << v[i] << " ";
13 }
```

Otro ejemplo de devolución de array por una función

Problema

Quitar los elementos consecutivos repetidos de un array (sin centinela), guardando el resultado en otro array.

Ejemplo de devolución de un array por una función

```
1  #include <iostream>
2  using namespace std;
3  /**
4   * Metodo para imprimir vector: nos aseguramos que el vector no se modificara
5   * @param vector a imprimir
6   * @param numero de elementos en el vector
7   */
8  void imprime_array(const char v[], int util);
9  /**
10  * Metodo para quitar repetidos: solo si son valores consecutivos
11  * @param vector original
12  * @param contador de elementos en el vector original
13  * @param vector de destino
14  * @return contador de elementos en el vector resultado
15  */
16 int quita_repes(const char original[], int util_original, char destino[]);
```

Ejemplo de devolución de un array por una función

```
1  // Quitar repetidos consecutivos
2  int quita_repes(const char original[], int util_original, char destino[]){
3    int util_destino=1;
4    // Se copia el primero tal cual
5    destino[0] = original[0];
6    // Bucle de recorrido del vector: desde la primera posicion
7    // en adelante. Se copia el valor si no es igual al previo
8    for (int i=1; i<util_original;i++){
9       if (original[i] != original[i-1]){
10         destino[util_destino] = original[i];
11         util_destino++;
12      }
13   }
14   // Se devuelve el contador de elementos
15   return util_destino;
16 }
```

Ejemplo de devolución de un array por una función

```
1  // Metodo de impresion
2  void imprime_array(const char v[], int util){
3      for (int i=0; i<util; i++){
4          cout << v[i] << " ";
5      }
6  }
7  int main(){
8      const int DIM =100;
9      char entrada[DIM]={'b','b','i','e','n','n','n'}, salida[DIM];
10     int util_entrada = 7, util_salida;
11
12     // Se quitan los repetidos
13     util_salida=quita_repes(entrada, util_entrada, salida);
14     // Se muestra el vector
15     imprime_array(salida, util_salida);
16 }
```

Ejemplo de devolución de un array por una función

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

Trabajando con arrays locales a funciones

Problema: Comprobar si un array de dígitos (0 a 9) de int es capicúo

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

Problema: Comprobar si un array de dígitos (0 a 9) de int es capicúo

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.

Cuestiones a considerar

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

Trabajando con arrays locales a funciones

Problema: Comprobar si un array de dígitos (0 a 9) de int es capicúo

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.

Cuestiones a considerar

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

Hay 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

Hay 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

Hay 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

Hay 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 la clase `vector` STL.

```
1 #include <iostream>
2 using namespace std;
3 const int DIM = 100;
4
5 void quita_nodigitos(const int original[],
6      int util_original,int destino[], int &util_destino);
7 void imprimevector(const int v[], int util);
8 bool capicua(const int v[], int longitud);
9
10 void imprimevector(const int v[], int util){
11    for (int i=0; i<util; i++)
12        cout << v[i] << " ";
13 }
```

```
1  void quita_nodigitos(const int original[],
2           int util_original, int destino[],
3           int &util_destino){
4      util_destino=0;
5      for (int i=0; i<util_original; i++)
6          if (original[i] > -1 && original[i] < 10){
7              destino[util_destino]=original[i];
8              util_destino++;
9          }
10 }
```

```
1  bool capicua(const int v[], int longitud){
2     bool escapicua = true;
3     int solodigitos[DIM];
4     int long_real;
5
6     quita_nodigitos(v, longitud, solodigitos, long_real);
7     for (int i=0; i< long_real/2 && escapicua; i++)
8        if(solodigitos[i] != solodigitos[long_real-1-i])
9           escapicua = false;
10    return escapicua;
11 }
```

```
1  int main(){
2      int entrada1[DIM]={1,2,3,4,3,2,1};
3      int util_entrada1=7;
4      int entrada2[DIM]={1,2,3,4,5,6,10, 7,8,9,10, 11, 9,12,
8, 13, 7, 6, -1, 5, 4, 3, 2, 1};
5      int util_entrada2=24;
6
7      imprimevector(entrada1, util_entrada1);
8      if (capicua(entrada1, util_entrada1))
9          cout << " es capicua\n";
10     else
11         cout << " no es capicua\n";
12     imprimevector(entrada2, util_entrada2);
13     if (capicua(entrada2, util_entrada2))
14         cout << " es capicua\n";
15     else
16         cout << " no es capicua\n";
17 }
```

Contenido del tema

1. Lo que sabemos hacer
2. Control de elementos usados de un array
   - Control del tamaño de un array con una variable
   - Control del tamaño de un array con un elemento centinela
3. Funciones y arrays
   - Paso de argumentos: array
   - Devolución de arrays por funciones
   - Trabajando con arrays locales a funciones
4. Cadenas de caracteres estilo C
5. Matrices de 2 dimensiones
   - Declaración e inicialización
   - Acceso, asignación, lectura y escritura
   - Sobre el tamaño de las matrices
   - Matrices de más de 2 dimensiones
6. Funciones y matrices
7. Gestión de filas de una matriz como arrays
8. Cuestiones abiertas

Cadenas de caracteres estilo C

Cadena de caracteres

Secuencia de caracteres de longitud variable.
ejemplos: apellidos, direcciones, id passaporte, DNI etc...

Tipos de cadenas de caracteres en C++

1. **cstring**: cadena de caracteres heredado de C.
2. **string**: cadena de caracteres propia de C++, es una clase.

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 *Control del tamaño de un array con un 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 contiene.
- Su tipo es un array de `char` con un tamaño reservado 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 contiene.
- Su tipo es un array de `char` con un tamaño reservado 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 I

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 }
```

Paso de cadenas a funciones II

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 las operaciones de lectura y escritura ya conocidas.

```
1 #include<iostream>
2 using namespace std;
3
4 int main(){
5     char nombre[80];
6     cout << "Introduce tu nombre: ";
7     cin >> nombre;
8     cout << "El nombre introducido es: " << nombre;
9 }
```

Entrada/salida de cadenas

Cuidado¡!

`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 se alcanza el límite de lectura.

Cuidado¡!

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

Entrada/salida de cadenas

```
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;
```

```
Introduce tu nombre: Sylvia Acid
El nombre introducido es: Sylvia Acid
Introduce tu edad: 121
La edad introducida es: 121
Introduce tu direccion: La direccion introducida es:
```

Entrada/salida de cadenas

Cuidado¡!

`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

Cuidado¡!

`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)

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

```
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
6 int main(){
7     char cadena1[]="Hola";
8     string cadena2;
9     char cadena3[10];
10
11    cadena2=cadena1; // cstring-->string
12    strcpy (cadena3, cadena2.c_str()); // string-->cstring
13    cout<<"cadena2="<<cadena2<<endl;
14    cout<<"cadena3="<<cadena3<<endl;
15 }
```

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.
- `const 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", c2[DIM];
7    strcpy(c2, "mundo");
8    strcat(c1, " ");
9    strcat(c1, c2);
10   cout <<"Longitudes:"<<strlen(c1)<<" "<<strlen(c2);
11   cout << "\nc1: " << c1 << " c2: " << c2;
12   if (strcmp(c1,"adiós mundo cruel") < 0)
13       cout << "\nCuidado con las mayúsculas\n";
14   if (strcmp(c2, "mucho") > 0)
15       cout << "\n\"mundo\" es mayor que \"mucho\"\n";
16 }
```

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  int main() {
8     const int DIM=100;
9     char cadena[DIM];
10
11    cout << "Introduce una cadena: ";
12    lee_linea(cadena, DIM);
13    deja_solo_primera_palabra(cadena);
14    cout << "Resultado = " << cadena << endl;
15 }
16 void deja_solo_primera_palabra(char c[])  {
17   int i=0;
18   // No hay espacios en blanco al inicio
19   while (c[i] != ' ' && c[i] != '\t' && i < strlen(c))
20     i++;
21   if (i < strlen(c))
22     c[i] = '\0';
23 }
```

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

Contenido del tema

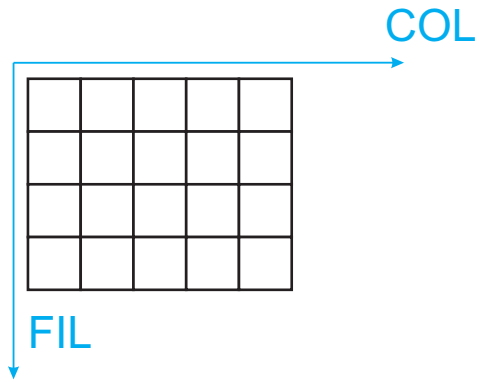
1. Lo que sabemos hacer
2. Control de elementos usados de un array
   - Control del tamaño de un array con una variable
   - Control del tamaño de un array con un elemento centinela
3. Funciones y arrays
   - Paso de argumentos: array
   - Devolución de arrays por funciones
   - Trabajando con arrays locales a funciones
4. Cadenas de caracteres estilo C
5. Matrices de 2 dimensiones
   - Declaración e inicialización
   - Acceso, asignación, lectura y escritura
   - Sobre el tamaño de las matrices
   - Matrices de más de 2 dimensiones
6. Funciones y matrices
7. Gestión de filas de una matriz como arrays
8. Cuestiones abiertas

Declaración

<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

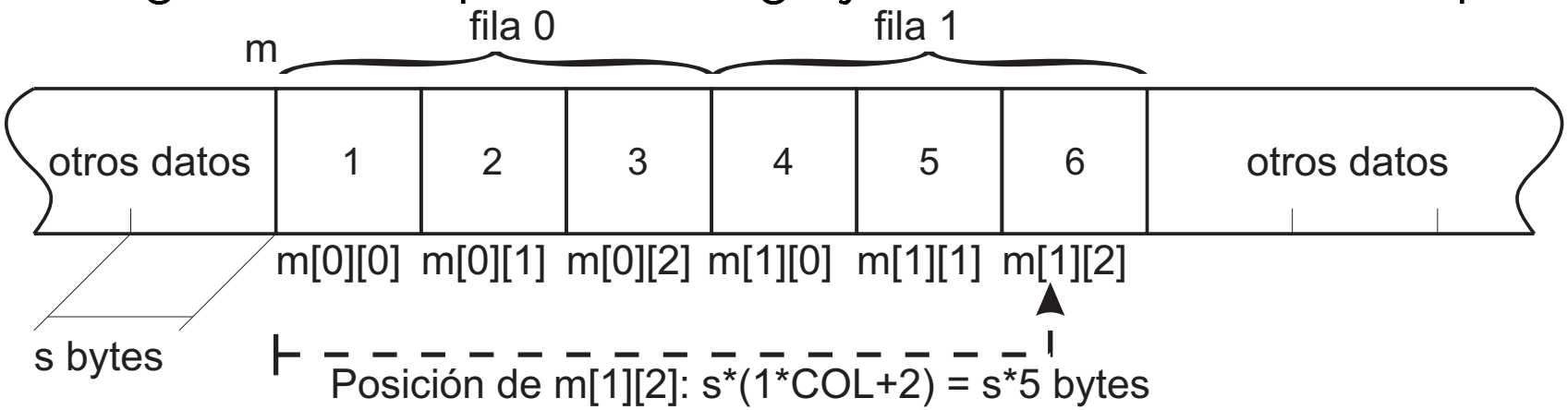
- El compilador procesa las matrices como array de arrays.
- Es decir, es un array con un tipo base también array (cada fila).
- En la declaración
  `int m[2][3]`
  `m` es un array de 2 elementos (`m[2]`) y cada elemento es un array de 3 `int` (`int xxxx[3]`).
- Observad que la sintaxis de la inicialización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

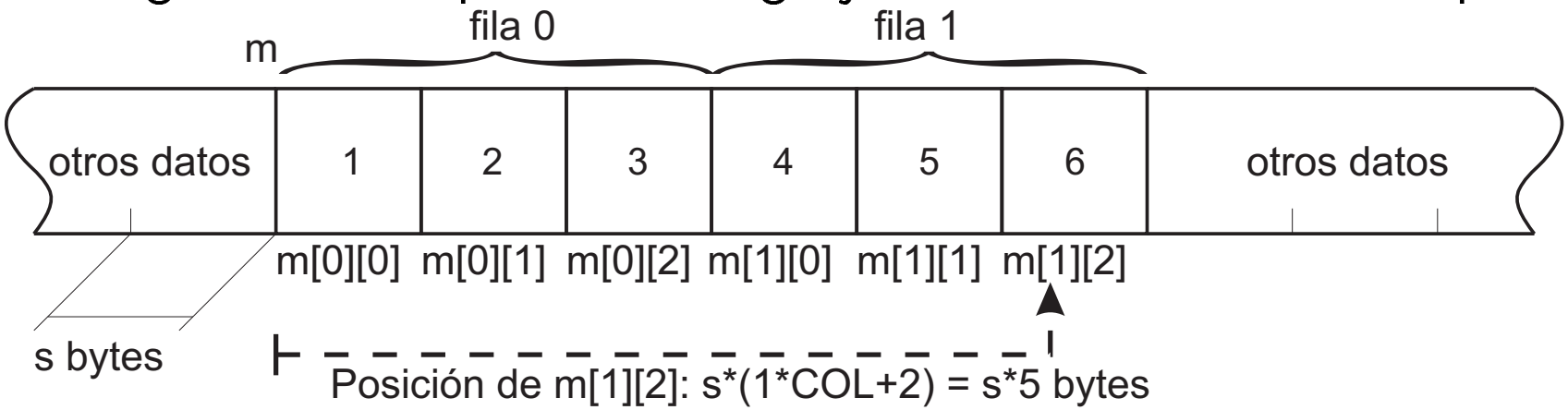


Almacenamiento en memoria de matrices

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.



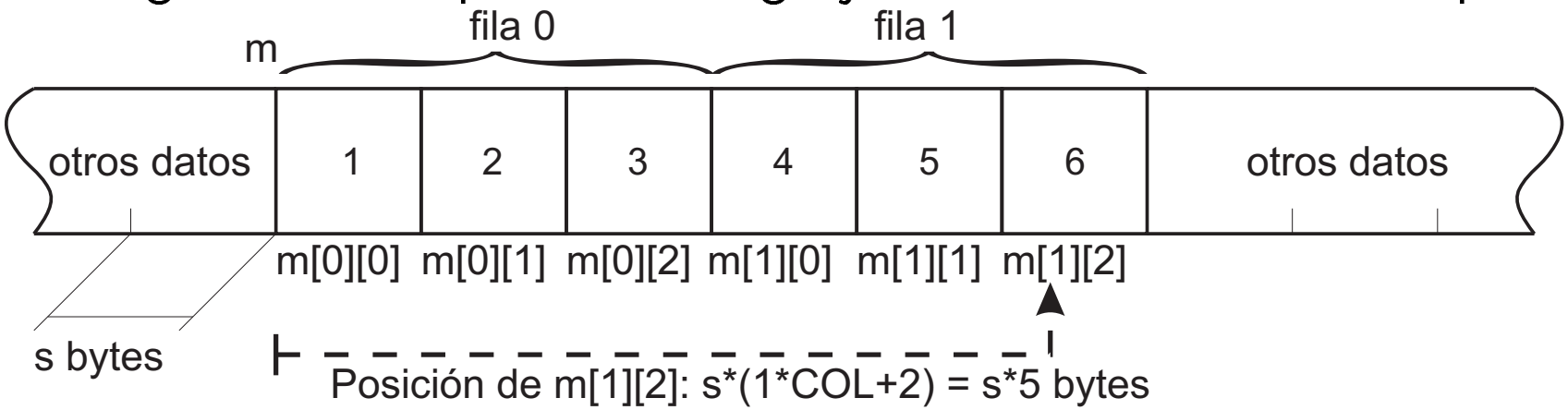
- Para acceder al elemento `m[i][j]` en una matriz `FIL` $\times$ `COL` el compilador debe *pasar a la fila* `i` y desde ahí moverse j elementos

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` × `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`

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>];
```

Sobre el tamaño de las matrices

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
10   do{
11     cout << "Introducir el número de filas: ";
12     cin >> util_fil;
13   }while ((util_fil<1) || (util_fil>FIL));
```

Sobre el tamaño de las matrices

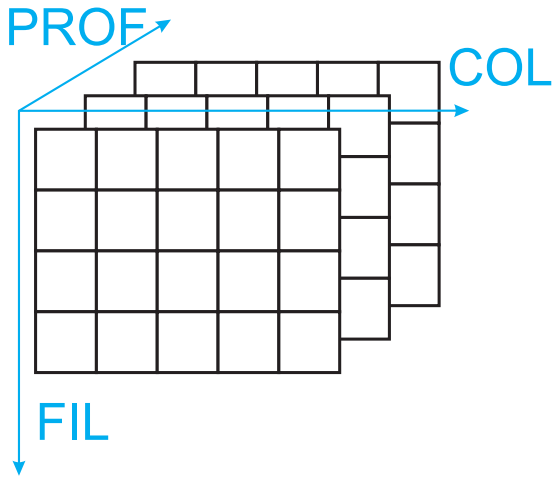
```
1   do{
2     cout << "Introducir el número de columnas: ";
3     cin >> util_col;
4   }while ((util_col<1) || (util_col>COL));
5
6   for (f=0 ; f<util_fil; f++)
7     for (c=0 ; c<util_col ; c++){
8       cout << "Introducir el elemento ("
9            << f << "," << c << "): ";
10      cin >> m[f][c];
11    }
12  cout << "\nIntroduzca elemento a buscar: ";
13  cin >> buscado;
```

Sobre el tamaño de las matrices

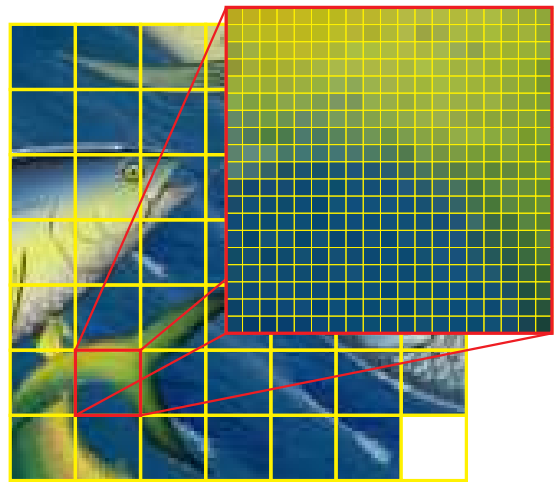
```
1   for (f=0; !encontrado && (f<util_fil) ; f++)
2     for (c=0; !encontrado && (c<util_col) ; c++)
3       if (m[f][c] == buscado){
4         encontrado = true;
5         fil_enc = f; col_enc = c;
6       }
7   if (encontrado)
8     cout << "Encontrado en la posición "
9       << fil_enc << "," << col_enc << endl;
10  else
11    cout << "Elemento no encontrado\n";
12
13  return 0;
14 }
```

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

1. Lo que sabemos hacer
2. Control de elementos usados de un array
   - Control del tamaño de un array con una variable
   - Control del tamaño de un array con un elemento centinela
3. Funciones y arrays
   - Paso de argumentos: array
   - Devolución de arrays por funciones
   - Trabajando con arrays locales a funciones
4. Cadenas de caracteres estilo C
5. Matrices de 2 dimensiones
   - Declaración e inicialización
   - Acceso, asignación, lectura y escritura
   - Sobre el tamaño de las matrices
   - Matrices de más de 2 dimensiones
6. Funciones y matrices
7. Gestión de filas de una matriz como arrays
8. Cuestiones abiertas

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  void lee_matriz(double m[][COL],
5                  int util_fil, int util_col){
6    for (int f=0 ; f<util_fil; f++)
7      for (int c=0 ; c<util_col ; c++){
8        cout << "Introducir el elemento ("
9             << f << "," << c << "): ";
10       cin >> m[f][c];
11     }
12 }
```

```
1  void busca_matriz(const double m[][COL], int util_fil,
2          int util_col, double elemento,
3          int &fil_encontrado, int &col_encontrado){
4    bool encontrado=false;
5    fil_encontrado = -1; col_encontrado = -1;
6    for (int f=0; !encontrado && (f<util_fil) ; f++)
7      for (int c=0; !encontrado && (c<util_col) ; c++)
8        if (m[f][c] == elemento){
9          encontrado = true;
10         fil_encontrado = f;
11         col_encontrado = c;
12       }
13 }
```

```
1  int lee_int(const char mensaje[], int min, int max){
2    int aux;
3    do{
4      cout << mensaje;
5      cin >> aux;
6    }while ((aux<min) || (aux>max));
7    return aux;
8  }
9  int main(){
10   double m[FIL][COL];
11   int fil_enc, col_enc, util_fil, util_col;
12   double buscado;
13
14   util_fil = lee_int("Introducir el número de filas: ",
15                      1, FIL);
16   util_col = lee_int("Introducir el número de columnas: ",
17                      1, COL);
18   lee_matriz(m, util_fil, util_col);
```

```
1    cout << "\nIntroduzca elemento a buscar: ";
2    cin >> buscado;
3
4    busca_matriz(m, util_fil, util_col, buscado,
5                 fil_enc, col_enc);
6    if (fil_enc != -1)
7      cout << "Encontrado en la posición "
8        << fil_enc << "," << col_enc << endl;
9    else
10     cout << "Elemento no encontrado\n";
11
12   return 0;
13 }
```

Contenido del tema

1. Lo que sabemos hacer
2. Control de elementos usados de un array
   - Control del tamaño de un array con una variable
   - Control del tamaño de un array con un elemento centinela
3. Funciones y arrays
   - Paso de argumentos: array
   - Devolución de arrays por funciones
   - Trabajando con arrays locales a funciones
4. Cadenas de caracteres estilo C
5. Matrices de 2 dimensiones
   - Declaración e inicialización
   - Acceso, asignación, lectura y escritura
   - Sobre el tamaño de las matrices
   - Matrices de más de 2 dimensiones
6. Funciones y matrices
7. Gestión de filas de una matriz como arrays
8. Cuestiones abiertas

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    int f;
5    fil_enc = -1;
6    col_enc = -1;
7    for (f=0; col_enc == -1 && (f<util_fil); f++)
8      col_enc = busca_sec(m[f], util_col, elemento);
9    if (col_enc !=-1)
10     fil_enc = f-1;
11 }
```

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   }
```

Gestión de filas de una matriz como arrays IV

```
11 }
```

Contenido del tema

1. Lo que sabemos hacer
2. Control de elementos usados de un array
   - Control del tamaño de un array con una variable
   - Control del tamaño de un array con un elemento centinela
3. Funciones y arrays
   - Paso de argumentos: array
   - Devolución de arrays por funciones
   - Trabajando con arrays locales a funciones
4. Cadenas de caracteres estilo C
5. Matrices de 2 dimensiones
   - Declaración e inicialización
   - Acceso, asignación, lectura y escritura
   - Sobre el tamaño de las matrices
   - Matrices de más de 2 dimensiones
6. Funciones y matrices
7. Gestión de filas de una matriz como arrays
8. Cuestiones abiertas

- Supongamos que necesitamos definir una función : `bool vacioSi()` con un array como parámetro pero para diferentes tipos, concretamente, para arrays de (`int / char /double`). ¿Es necesario implementar una función para cada tipo de array?
- Supongamos que necesitamos definir una función : bool vacioSi() con un array como parámetro que vaya a ser utilizado con arrays de diferentes longitudes. ¿Es necesario implementar una función para varias longitudes?
- dado un vector tipo double *v*, con *util_v* componentes y capacidad *CAPACIDAD*
  1. escribe la cabecera de *buscarElementoEnVector*
  2. escribe la cabecera de *ordenarCreciente*
  3. escribe la cabecera de *aniadir* elemento
- Cómo se pasan arrays como parámetro de E/ ¿?
- Cómo se pasan arrays como parámetro de E/S ¿?

- ¿Todo array char es una cadena de C?
- ¿Toda cadena C es un array char ?
- Enumera al menos 3 diferencias entre cadena C y string
- Qué se puede hacer con una cadena C y que no se pueda hacer con un array de char?
- En un vector de 100 elementos de tipo char, **simbolos**, que contiene 10 elementos válidos. Cómo se accede a la última componente válida del vector¿? Existe alguna discrepancia entre la capacidad efectiva de un vector y el espacio total reservado¿?
- sean nombre y DNI dos cadenas ya rellenas
  1. escribe la funcion **concatena()** con cadena destino y origen
  2. escribe el main para concatenar nombre y DNI.
  3. compara la cabecera de concatena con **strcat()**


creative
commons


BY NC SA