2. Matrices con dimensiones variables acotadas

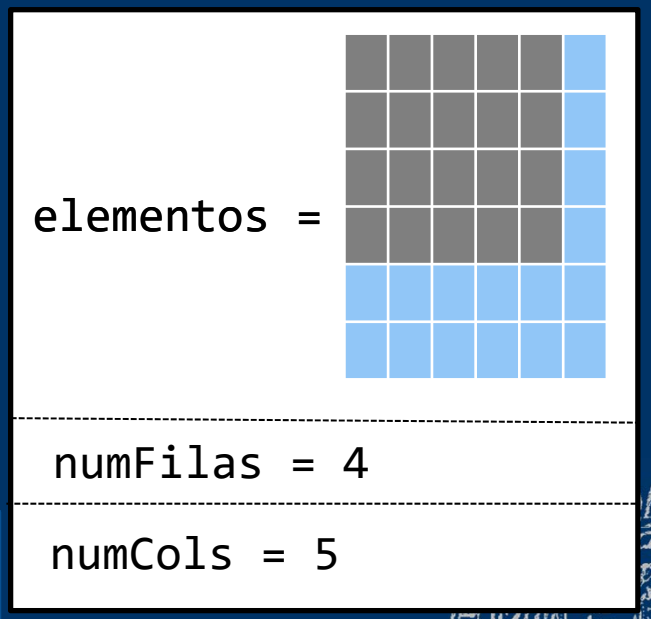
Arrays bidimensionales

Matrices con dimensiones variables acotadas

```
const int MAX_DIM = 6; // Tamaño máximo estimado > 0

typedef struct {
  int numFilas, numCols;
  double elementos[MAX_DIM][MAX_DIM];
} tMatriz;
```

```
tMatriz matriz;
matriz.numFilas = 4;
matriz.numCols = 5;
```

matriz =

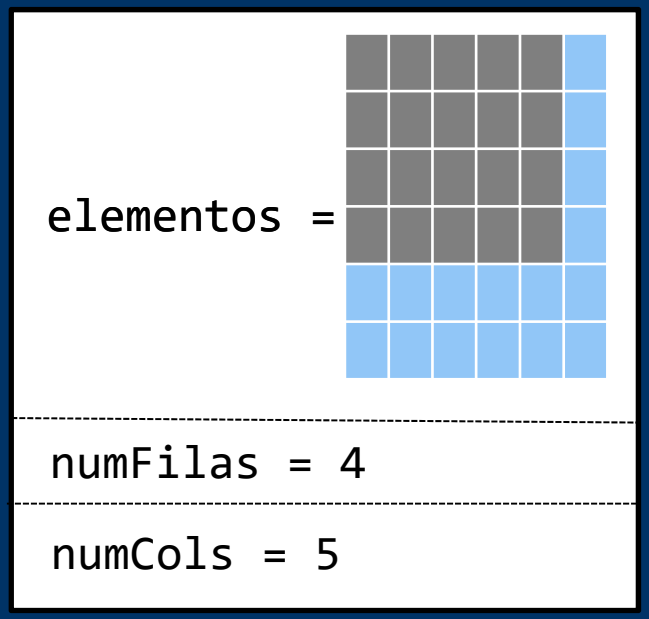


Las estructuras se pasan por valor (sin &)
o por referencia (con &) a los subprogramas

Arrays bidimensionales

Matrices con dimensiones variables acotadas

```
tMatriz matriz;
matriz.numFilas = 4;
matriz.numCols = 5;
```

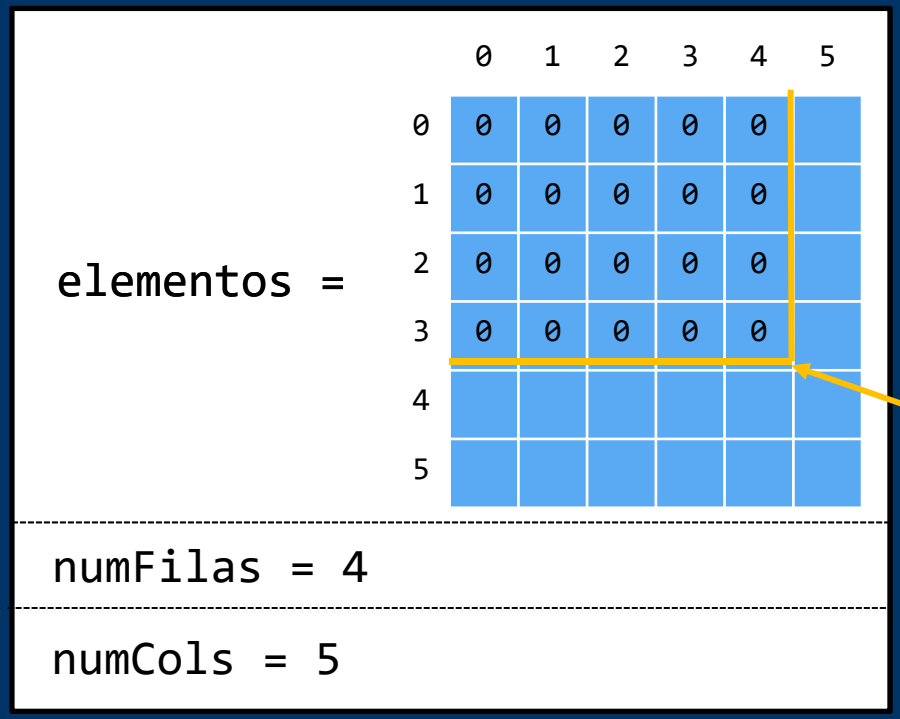



```
// Parte ocupada: [0..matriz.nFilas)x[0..matriz.numCols)
  for (int fila = 0; fila < matriz.numFilas; ++fila)
    for (int col = 0; col < matriz.numCols; ++col)
      matriz.elementos[fila][col] = 0;
```

Arrays bidimensionales

```
void iniciar(tMatriz & mat, int nf, int nc, double vi) {
   mat.numFilas = nf; mat.numCols = nc;
   for (int fila = 0; fila < mat.numFilas; ++fila)
     for (int col = 0; col < mat.numCols; ++col)
       mat.elementos[fila][col] = vi;
}
iniciar(matriz,4,5,0);
```

matriz =



numCols es 5 y MAX_DIM es 6

numFilas es 4 y MAX_DIM es 6

Parte ocupada: [0..nF)x[0..nC)

Arrays bidimensionales

```
… recorrer(tMatriz /*const*/&/*ent/sal?*/ mat …) {
   for (int fila = 0; fila < mat.numFilas; ++fila)
     for (int col = 0; col < mat.numCols; ++col)
        // Procesar mat.elementos[fila][col];
}
```

```
typedef struct {int fila; int col;} tCoor;

bool buscar(tMatriz const& mat, tCoor & pos) {
     bool enc = false; pos.fila = 0;
     while (pos.fila < mat.numFilas && !enc) {
      pos.col = 0;
      while (pos.col < mat.numCols && !enc)
        if (prop(mat.elementos[pos.fila][pos.col]))
              enc = true;
        else ++pos.col;
      if (!enc) ++pos.fila;
     }
     return enc;
 }
```

Si el elemento cumple la propiedad

Parámetro de entrada: const&

Parámetro de salida o ent/sal: &

Arrays bidimensionales

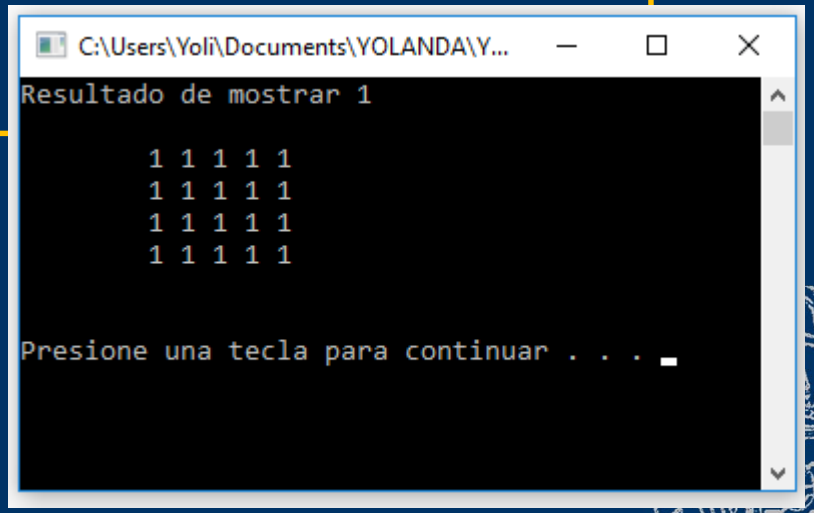
Ejemplo:

Mostrar matriz por filas (con una columna singular)

Arrays bidimensionales

Ejemplo: Mostrar matriz por filas (con una columna singular)

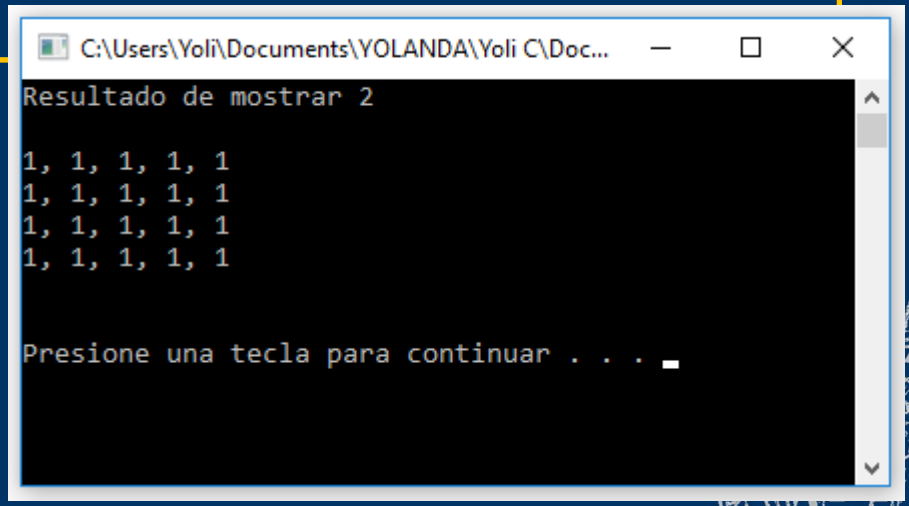
```
void mostrar1(tMatriz const& mat) {
  for (int fila = 0; fila < mat.numFilas; ++fila) {
    cout << '\t' << mat.elementos[fila][0];
    for (int col = 1; col < mat.numCols; ++col)
      cout << ', ' << mat.elementos[fila][col];
    cout << '\n';
  }
}
```



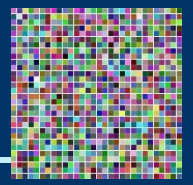
Arrays bidimensionales

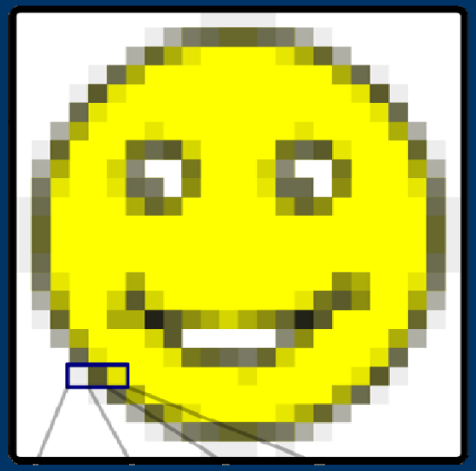
Ejemplo: Mostrar matriz por filas (con una columna singular)

```
void mostrar2(tMatriz const& mat) {
  for (int fila = 0; fila < mat.numFilas; ++fila) {
    for (int col = 0; col < mat.numCols-1; ++col)
      cout << mat.elementos[fila][col] << ', ';
    cout << mat.elementos[fila][mat.numCols-1] << '\n';
  }
}
```



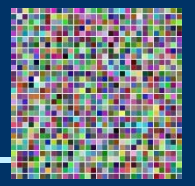
Ejemplo: Imagen (bmp)





R 93%	R 35%	R 90%
G 93%	G 35%	G 90%
B 93%	B 16%	B 0%

Ejemplo: Imagen (bmp)



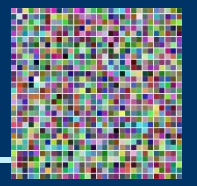
```
typedef unsigned int uint; // entero (32 bits) sin signo
typedef unsigned short int usint; // entero pequeño sin signo

typedef struct {
        usint rojo;
        usint verde;
        usint azul;
} tRGB;
```

```
tRGB pixel;
```

pixel =

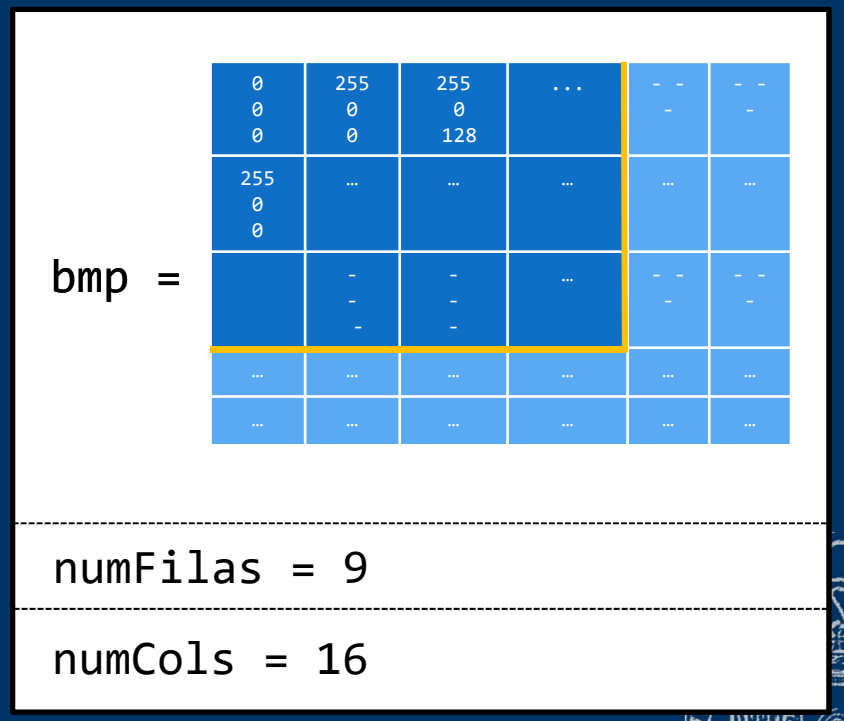
rojo =	255
verde =	0
azul=	128

Ejemplo: Imagen (bmp)



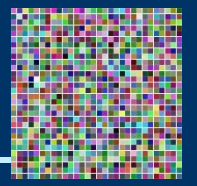
```
const usint Max_Res = 24;    // máximo nº de filas y columnas

typedef struct {
  usint numFilas, numCols;
  tRGB bmp[Max_Res][Max_Res]; // Max_Res*Max_Res pixeles
} tImagen;

tImagen imagen;
imagen.numFilas = 9;
imagen.numCols = 16;
imagen.bmp
```

imagen =



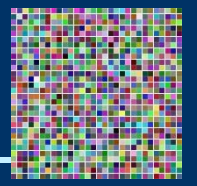

```
// los struct se pueden asignar
imagen.bmp[0][2] = pixel;
```

Ejemplo: Operadores con pixeles



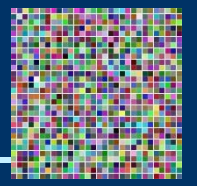
```
// operador de comparación de igualdad
// Son iguales si tienen la misma cantidad de rojo, verde y azul
bool operator == (tRGB const& c1, tRGB const& c2) {
  return c1.azul == c2.azul && c1.rojo == c2.rojo
         && c1.verde == c2.verde;
}

// operador de comparación de desigualdad
bool operator != (tRGB const& c1, tRGB const& c2) {
  return !(c1 == c2);
}
```

Ejemplo: mostrar pixeles



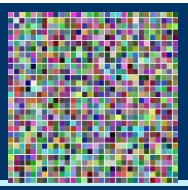
```
void mostrar(tRGB const& color) {
  cout << "Rojo: " << usint(color.rojo)  << '\n';
  cout << "Verde: "<< usint(color.verde) << '\n';
  cout << "Azul: " << usint(color.azul)  << endl;
}
```

Ejemplo: Girar una imagen

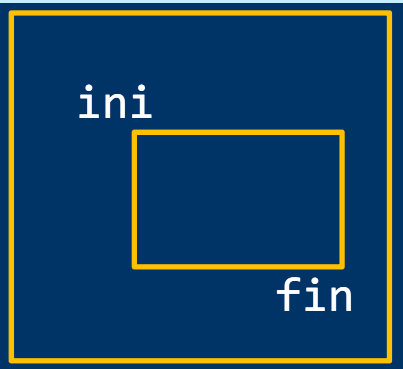


```
void girar(tImagen &/*ent/sal*/ imagen) {
  tImagen aux = imagen; // los struct se pueden asignar
  imagen.numFilas = aux.numCols;
  imagen.numCols = aux.numFilas;
  for (usint f = 0; f < aux.numFilas; ++f)
    for (usint c = 0; c < aux.numCols; ++c)
      // traspuesta
        imagen.bmp[c][f] = aux.bmp[f][c];

      // giro derecha
      // imagen.bmp[c][aux.numFilas-1 - f] = aux.bmp[f][c];
      // giro izquierda
      // imagen.bmp[aux.numCols-1 - c][f] = aux.bmp[f][c];
}
```

Ejemplo: Rellenar una imagen



submatriz




```
typedef struct {
    int fila;
    int col;
} tCoor;

void rellenar(tImagen & imagen, tRGB const & pixel){
  rellenar(imagen, {0, 0},
                   {imagen.numFilas, imagen.numCols}, pixel);
}

void rellenar(tImagen & imagen, tCoor const& ini,
                                tCoor const& fin, tRGB const& pixel){
  for (usint f = ini.fila; f < fin.fila; ++f)
    for (usint c = ini.col; c < fin.col; ++c)
      imagen.bmp[f][c] = pixel;
}
```


Arrays bidimensionales. Volvemos a tMatriz

Arrays bidimensionales. Volvemos a tMatriz

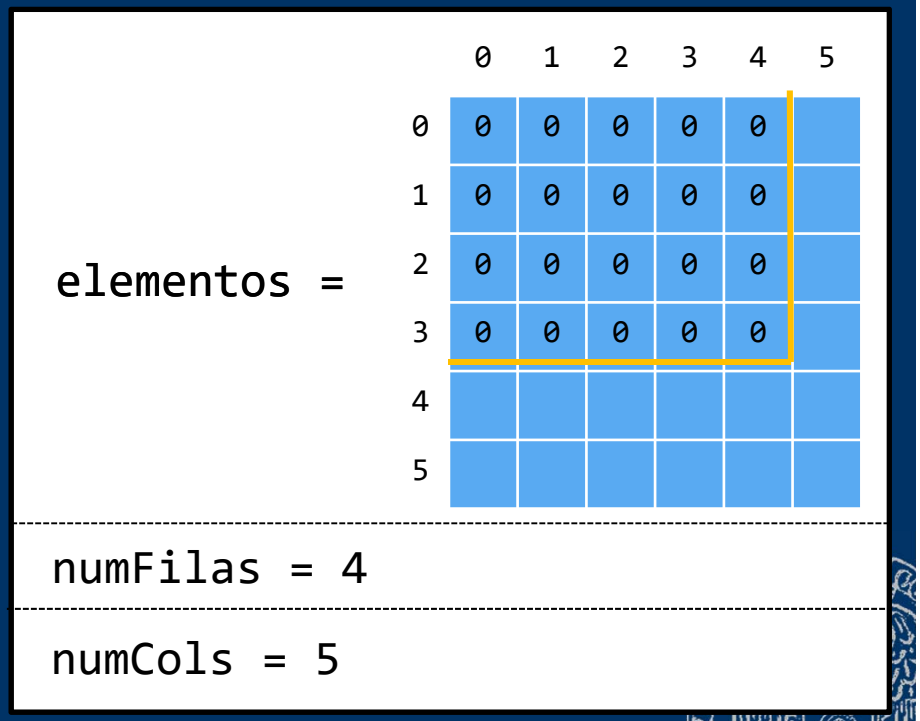
Búsqueda del primer elemento que cumple una propiedad

```
const int MAX = …;
typedef struct {
  int numFilas, numCols;
  tElem elementos[MAX][MAX];
} tMatriz;
```

```
typedef struct {
  int fila;
  int col;
} tCoor;
```

```
tMatriz matriz;
matriz.numFilas = 4;
matriz.numCols = 5;
```

matriz =



Arrays bidimensionales

Versión 1

Búsqueda del primer elemento que cumple una propiedad

Localizar la posición:

```
bool buscar(tMatriz const& mat, tCoor & pos) {
  bool enc = false;
  pos.fila = 0;
  while (pos.fila < mat.numFilas && !enc) {
      pos.col = 0;
      while (pos.col < mat.numCols && !enc)
         if (prop(mat.elementos[pos.fila][pos.col],…))
                enc = true;
         else ++pos.col;
      if (!enc) ++pos.fila;
  }
  return enc;
}
```

Arrays bidimensionales

Versión 2

Búsqueda del primer elemento que cumple una propiedad

```
bool buscar(tMatriz const& mat, tCoor & pos) {
  bool enc = false; pos.fila = 0;
  while (pos.fila < mat.numFilas && !enc) {
    enc = buscar(mat, pos.fila, pos.col); // buscar en una fila
    if (!enc) ++pos.fila;
  }
  return enc;
}
```

```
bool buscar(tMatriz const& mat, int fila, int & col) {
   bool enc = false; col = 0;
   while (col < mat.numCols && !enc)
     if (prop(mat.elementos[fila][col],…)) enc = true;
     else ++col;
   return enc;
}
```

Arrays bidimensionales

Búsqueda del primer elemento que cumple una propiedad desde una determinada posición

```
tMatriz matriz;
tCoor pos;

bool buscar(tMatriz const& mat, tCoor & pos);
// no es necesario inicializar pos

bool buscarDesde(tMatriz const& mat,
                 tCoor & pos);

// pos tiene que tener un valor a partir del cual se buscará

// La función devuelve true si lo encuentra
// pos almacena la posición del elemento encontrado
```

Arrays bidimensionales

Búsqueda desde una posición

```
bool buscarDesde(tMatriz const& mat, tCoor&/*ent/sal*/ pos){
  bool enc = false;  // pos ya tiene un valor
  int col = pos.col;
  while (pos.fila < mat.numFilas && !enc) {
    while (pos.col < mat.numCols && !enc)
      if (prop(mat.elementos[pos.fila][pos.col],…))
            enc = true;
      else ++pos.col;
    if (!enc) {++pos.fila; pos.col = col; }
  }
  return enc; // pos puede haber cambiado de valor
}
```

Arrays bidimensionales

Ejemplo:

Averiguar si una matriz cuadrada es triangular inferior
(los elementos por encima de su diagonal principal son cero)

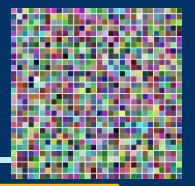
```
diagonal principal: col == fila

M[f][c] = 0 para todo c > f
```

Arrays bidimensionales

Ejemplo: Averiguar si una matriz cuadrada es triangular inferior (los elementos por encima de su diagonal principal son cero)

```
bool triangularInf(tMatriz const& mat) {
  bool tri = mat.numFilas == mat.numCols; // cuadrada?
  int fila = 0; int col;
  while (fila < mat.numFilas && tri) { // por filas
    col = fila + 1;
    while (col < mat.numCols && tri) {
      if (mat.elementos[fila][col] != 0.0) tri = false;
      else ++col;
    }
    if (tri) ++fila;
  }
  return tri;
}                    // diagonal principal: col == fila
```

Ejemplo: Buscar imagen 3x3 en imagen

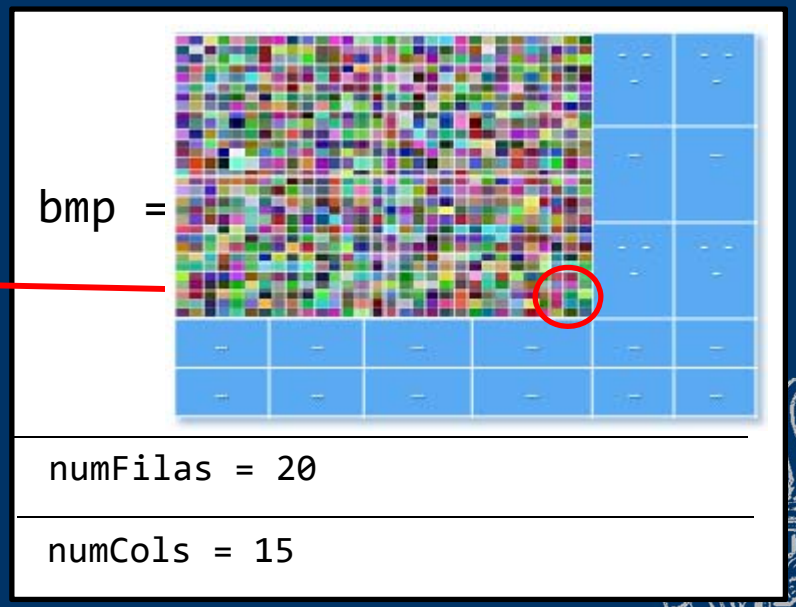


```
typedef tRGB tImg3x3[3][3]; // matriz 3x3

const usint Max_Res = 24;    // máximo nº de filas y columnas

typedef struct {
  usint numFilas, numCols;
  tRGB bmp[Max_Res][Max_Res]; // Max_Res*Max_Res pixeles
} tImagen;
```

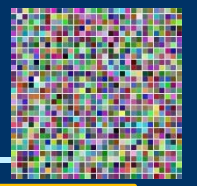
```
tImagen imagen;
tImag3x3 mat;
```

imagen

mat



Ejemplo: Buscar imagen 3x3 en imagen



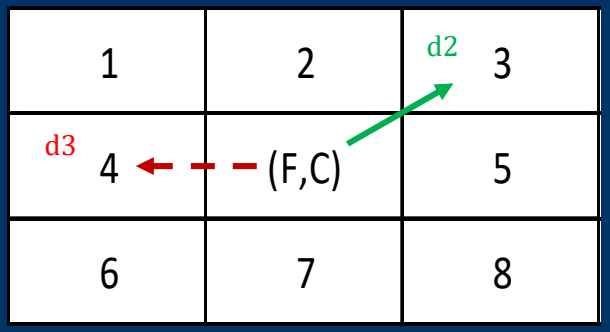
```
bool submatriz(tImagen const& imagen, tImg3x3 const mat,
               tCoor & pos) {

  bool enc = false;
  pos.fila = 0;
  usint filas = imagen.numFilas-2, cols = imagen.numCols-2;
  while (pos.fila < filas && !enc) {
    pos.col = 0;
    while (pos.col < cols && !enc)
      if (iguales3x3(imagen, pos, mat)) enc = true;
      else ++pos.col;
    if (!enc) ++pos.fila;
  }
  return enc;
}
```

Ejemplo: Buscar imagen 3x3 en imagen

```
bool iguales3x3(tImagen const& imagen, tCoor const& pos,
                tImg3x3 const mat) {
  bool iguales = true;
  usint f = 0, c;
  while (f < 3 && iguales) {
    c = 0;
    while (c < 3 && iguales)
      if (imagen.bmp[pos.fila + f][pos.col + c] != mat[f][c])
          iguales = false;
      else ++c;
    if (iguales) ++f;
  }
  return iguales;
}
```

Arrays bidimensionales

Recorrido de los elementos vecinos a uno dado (pos): submatriz 3x3 centrada en ese elemento




```
typedef struct {int fila; int col;} tCoor;
const int incF[] = {-1, -1, -1,  0, 0,  1, 1, 1};
const int incC[] = {-1,  0,  1, -1, 1, -1, 0, 1}; // 8 dirs.

const int NumDirs = 8;
```

d2 d3

Arrays bidimensionales

Recorrido de los elementos vecinos a uno dado (pos): submatriz 3x3 centrada en ese elemento

```
// dada una posición y una dirección (0-7), calcular la
posición // vecina en dicha dirección: dir

void vecina(tCoor const& pos, int dir, tCoor & vec){
  vec.fila = pos.fila + incF[dir];
  vec.col = pos.col + incC[dir];
}
```

Arrays bidimensionales

Recorrido de los elementos vecinos a uno dado (pos): submatriz 3x3 centrada en ese elemento

```
void recorrerVecinas(tMatriz & mat, tCoor const& pos,…){
  tCoor vec;
  for (int dir = 0; dir < NumDirs; ++dir) {
    vecina(pos, dir, vec);
      …
      // procesar mat[vec.fila][vec.col]
      // procesar(mat, vec);
  }
}
```

Arrays bidimensionales

Recorrido de diagonales en una matriz cuadrada NxN:

Diagonal principal: N elementos con fila == col (k==0)

N-1 diagonales superiores (k: 1...N-1): N-k elementos con col == fila + k

N-1 diagonales inferiores (k: 1...N-1): N-k elementos con fila == col + k

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

Diagonal k:
col = fila + k
elemento[fila][fila+k]
Empezar en fila=0

Diagonal k:
fila = col + k
elemento[col+k][col]
Empezar en col=0

Arrays bidimensionales

Ejemplo:

Calcular la suma de los elementos de una diagonal dada

Arrays bidimensionales

Ejemplo: Suma de los elementos de una diagonal

```
double sumarDiagonal(tMatriz const & mat, int k) {
    double cont = 0;
    if (k > 0) {// diagonal superior
        for (int col = k; col < mat.numCols; ++col) {
                cont += mat.elementos[col - k][col];
        }
    }else if (k < 0) {// diagonal inferior
        for (int fila = -k; fila < mat.numFilas; ++fila) {
                cont += mat.elementos[fila][fila + k];
        }
    }else { // diagonal principal: k==0 (col == fila)
        for (int fila = 0; fila < mat.numFilas; ++fila) {
                cont += mat.elementos[fila][fila];
        }
    }
    return cont;
}
```

Arrays bidimensionales

Ejemplo:

Decidir si una matriz cuadrada es triangular inferior (por diagonales)

Arrays bidimensionales

Ejemplo: matriz cuadrada triangular inferior (por diagonales)

```
bool triangularInf(tMatriz const& mat) { // por diagonales
  bool tri = mat.numFilas == mat.numCols; // cuadrada?
  int k = 1, fila;
  while (k < mat.numCols && tri){ // diagonal k: col == fila + k
    fila = 0;
    while (fila < mat.numFilas - k && tri) {
      if (mat.elementos[fila][fila + k] != 0.0) tri = false;
      else ++fila;
    }
    if (tri) ++k;
  }
  return tri;
}      // diagonal principal: k==0 (col == fila)
```