

# Arrays de dimensión variable acotada

---

## 3. Arrays con dimensión variable acotada



# Arrays de dimensión variable acotada

*Array con un máximo de elementos + Contador de elementos*

```
const int TM = ...; // Tamaño máximo estimado: TM > 0  
typedef tDato tArray[TM]; // Por ejemplo tDato= float
```

El array y el contador están mutuamente relacionados ->

usamos una estructura para encapsularlos:

```
typedef struct {  
    int contador;  
    tArray datos; } tLista;  
tLista lista;
```

Parte ocupada: 0..contador-1



# Arrays de dimensión variable acotada

```
const int MAX = 100; // Tamaño estimado > 0
typedef struct {
    int cont; // Primer índice libre <-> nº de elementos
    tDato datos[MAX]; // Hasta 100 elementos
} tLista; // Parte ocupada de 0 a (cont-1), resto libre
tLista lista;
lista.cont = 0; // lista vacía
```

Recorrido de la lista: De los elementos de la parte ocupada

Parámetro de entrada: const&  
Parámetro de salida o ent/sal: &

```
void recorrer(tLista const & lista, ...) {
    for (int i = 0; i < lista.cont; ++i)
        // Procesar lista.datos[i]
}
```




# Arrays de dimensión variable acotada

```
const int MAX = 100; // Tamaño estimado > 0
typedef struct {
    int cont; // Primer índice libre <-> nº de elementos
    tDato datos[MAX]; // Hasta 100 elementos
} tLista; // Parte ocupada de 0 a (cont-1), resto libre
tLista lista; lista.cont = 0; // lista vacía
```

Búsqueda en la lista: En la parte ocupada

```
bool buscar(tLista const& lista, ..., int & pos) {
    pos = 0; bool enc = false;
    while (pos < lista.cont && !enc)
        if (... lista.datos[pos]... ) enc = true; // encontrado en pos
        else ++pos;
    return enc;
}
```



# Arrays de dimensión variable acotada

Insertar nuevo último (push\_back)

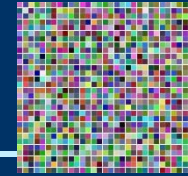
```
bool push_back(tLista & lista, tDato const& dato) {  
    if (lista.cont == MAX) return false; // lista llena  
    else {  
        lista.datos[lista.cont] = dato;  
        lista.cont += 1;  
        return true;  
    }  
}
```

Eliminar el último (pop\_back)

```
bool pop_back(tLista & lista) {  
    if (lista.cont == 0) return false; // lista vacía  
    else { lista.cont -= 1; return true; }  
}
```



# Ejemplo: Histograma de una imagen

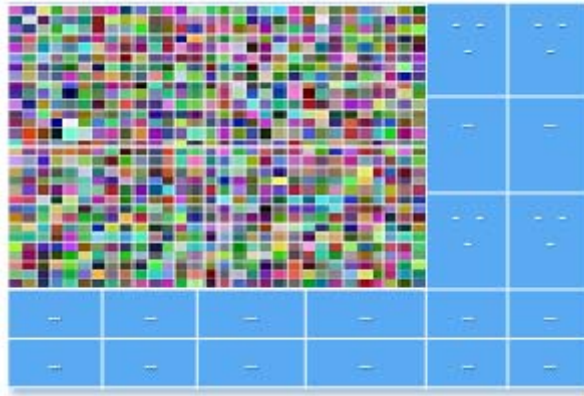


*Ejemplo:*

*Construir el histograma asociado a una imagen*

imagen

bmp =



numFilas = 20

numCols = 15

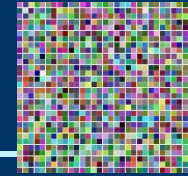
histograma

numColores = 3

frecuencias =

pixel =	rojo = 255	pixel =	rojo = 0	pixel =	rojo = 255		...	
	verde = 0		verde = 0		verde = 0			
	azul= 128		azul= 0		azul= 15			
	cont = 83		cont = 21		cont = 40			

# Ejemplo: Histograma de una imagen



```
const uint MaxColores = Max_Res*Max_Res; // 24*24 = 576 colores
```

```
typedef struct {
```

```
    tRGB pixel;
```

```
    uint cont;
```

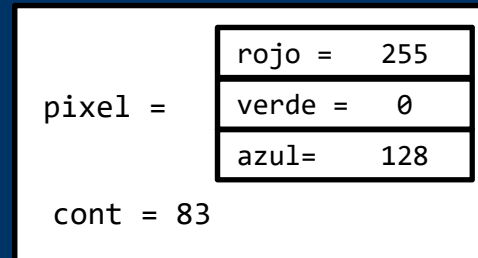
```
} tColorCont;
```

```
typedef struct {
```

```
    uint numColores; // parte ocupada 0..numColores-1
```

```
    tColorCont frecuencias[MaxColores];
```

```
} tHistograma;
```

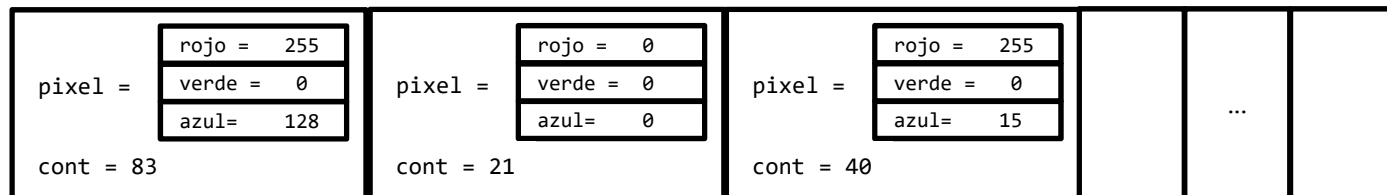


```
tHistograma histograma;
```

histograma

```
numColores = 3
```

```
frecuencias =
```



# Ejemplo: Histograma de una imagen



```
bool buscar(tHistograma const& histograma, tRGB const& pixel, uint /*sal*/ pos) {  
    pos = 0;  
    while (pos < histograma.numColores && histograma.frecuencias[pos].pixel != pixel)  
        ++pos; // != sobrecargado  
    return pos < histograma.numColores;  
}
```

```
void insertar(tHistograma /*ent/sal*/ histograma, tRGB const& pixel) {  
    uint pos;  
    if (buscar(histograma, pixel, pos)) histograma.frecuencias[pos].cont += 1;  
    else {  
        push_back(histograma, { pixel, 1 }); // sabemos que cabe  
    }  
}
```

```
void histogramaImagen(tImagen const& imagen, tHistograma /*sal*/ histograma) {  
    histograma.numColores = 0;  
    for (uint f = 0; f < imagen.numFilas; ++f)  
        for (uint c = 0; c < imagen.numCols; ++c)  
            insertar(histograma, imagen.bmp[f][c]);  
}
```





# Ejemplo: Histograma de una imagen

---



Ejemplo:

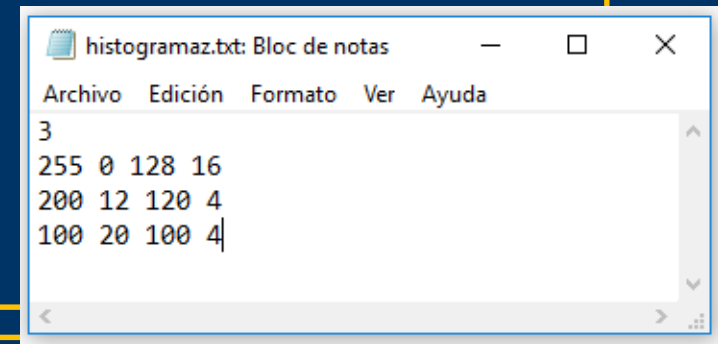
*Guardar el histograma en un fichero de texto*



# Ejemplo: Histograma de una imagen



```
bool guardar(tHistograma const& histograma, string const& nombArch) {
    ofstream flujo;
    flujo.open(nombArch);
    if (flujo.is_open()) {
        flujo << histograma.numColores << '\n'; // número de elementos
        for (usint i = 0; i < histograma.numColores; ++i) {
            flujo << histograma.frecuencias[i].pixel << ' '; // sobrecargado
            flujo << histograma.frecuencias[i].cont << '\n';
        }
        flujo.close();
        return true;
    } else return false;
}
```



```
ostream & operator <<(ostream & /*ent/sal*/ flujo, tRGB const& pixel) {
    flujo << uint(pixel.rojo) << ' ' << uint(pixel.verde) << ' '
        << uint(pixel.azul);
    return flujo;
}
```



# Ejemplo: Histograma de una imagen

---

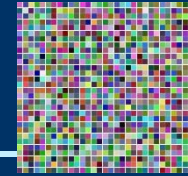


Ejemplo:

*Leer un histograma de un fichero de texto*



# Ejemplo: Histograma de una imagen

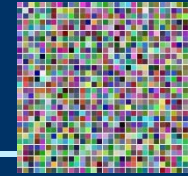


```
bool cargar(tHistograma & /*sal*/ histograma, string const& nombArch) {
    ifstream flujo;
    flujo.open(nombArch);
    if (flujo.is_open()) {
        flujo >> histograma.numColores;
        for (uint i = 0; i < histograma.numColores; ++i) {
            flujo >> histograma.frecuencias[i].pixel; // sobrecargado
            flujo >> histograma.frecuencias[i].cont;
        } flujo.close();
        return true;
    } else return false;
}

istream & operator >>(istream & /*ent/sal*/ flujo, tRGB & /*sal*/ pixel) {
    uint aux;
    flujo >> aux; pixel.rojo = uint8(aux);
    flujo >> aux; pixel.verde = uint8(aux);
    flujo >> aux; pixel.azul = uint8(aux);
    return flujo;
}
```



# Ejemplo: Histograma de una imagen



...

```
int main() {  
    tImagen imagen = { 9, 16, { // resto 0 !!  
        {{255,0,0},{255,0,0},{255,0,0},{0,255,0},{0,255,0},{0,255,0},{0,0,255},{0,0,255},{0,0,255}},  
        {{255,0,0},{255,0,0},{255,0,0},{0,255,0},{0,255,0},{0,255,0},{0,0,255},{0,0,255},{0,0,255}},  
        {{255,0,0},{255,0,0},{255,0,0},{0,255,0},{0,255,0},{0,255,0},{0,0,255},{0,0,255},{0,0,255}},  
        {{255,0,0},{255,0,0},{255,0,0},{0,255,0},{0,255,0},{0,255,0},{0,0,255},{0,0,255},{0,0,255}},  
        {{255,0,0},{255,0,0},{255,0,0},{0,255,0},{0,255,0},{0,255,0},{0,0,255},{0,0,255},{0,0,255}},  
        {{255,0,0},{255,0,0},{255,0,0},{0,255,0},{0,255,0},{0,255,0},{0,0,255},{0,0,255},{0,0,255}},  
        {{255,0,0},{255,0,0},{255,0,0},{0,255,0},{0,255,0},{0,255,0},{0,0,255},{0,0,255},{0,0,255}},  
        {{255,0,0},{255,0,0},{255,0,0},{0,255,0},{0,255,0},{0,255,0},{0,0,255},{0,0,255},{0,0,255}},  
        {{255,0,0},{255,0,0},{255,0,0},{0,255,0},{0,255,0},{0,255,0},{0,0,255},{0,0,255},{0,0,255}}  
    } };  
    tHistograma histograma;  
    histogramaImagen(imagen,histograma);  
    mostrar(histograma); // histograma.numColores == 4  
    return 0;           // .frecuencias == { {{255,0,0}, 27},  
    }                   {{0,255,0}, 27}, {{0,0,255}, 27}, {{0,0,0}, 63}, ...}
```

