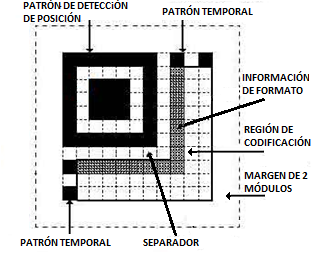
**Fundamentos de Programación (8 de septiembre de 2015)**

**(Grados en Ingeniería Mecánica, Eléctrica, Electrónica Industrial y Química Industrial)**

Construir un programa en ***C*** lo más modular posible (atendiendo a los criterios de modularidad) que se ajuste a la especificación dada y documentar el diseño preliminar con la definición de las nuevas tipologías de datos, el diagrama de módulos (estructura del programa) y las interfaces de los módulos, y el diseño detallado con las definiciones de los respectivos sub-programas.

**Ejercicio:** Construir un programa que lea una imagen en escala de grises escaneada por una cámara digital, e indique si la misma contiene algún ***micro código QR*** versión ***M1***, imprimiéndolos en pantalla en caso afirmativo. La imagen digital está formada por un cuadrado de ***64x64*** píxeles, donde cada píxel está representado por un valor real entre ***0.00*** (negro) y ***1.00*** (blanco) correspondiente a la intensidad de color blanco, y cuyos valores se encuentran almacenados en un archivo de texto ("***escaner001.txt***"), separados por al menos un carácter de espaciado y ordenados por filas de izquierda a derecha y de arriba hacia abajo.

El ***micro código QR*** es una versión más pequeña del estándar del ***código QR*** ("Quick Response": Respuesta Rápida) el cual fue creado en 1994 por la compañía japonesa ***Denso Wave***, subsidiaria de ***Toyota***, para almacenar información en una matriz de puntos o en un código de barras bidimensional (***ISO/IEC18004***). El ***código micro QR*** está adaptado para aplicaciones que requieren un espacio más pequeño y usan pequeñas cantidades de datos, tales como identificación de placas de circuito impresos y componentes electrónicos. La eficacia de la codificación de los datos se ve incrementada por el uso de sólo un patrón de detección de posición y de un margen de dos módulos alrededor del símbolo (frente a los tres patrones y cuatro módulos que requiere el código ***QR***).



La versión ***M1*** está formada por un cuadro de ***11x11*** módulos, delimitada por un margen de ***2*** módulos.

***Nota***: para la detección del ***micro código QR*** versión ***M1***, el programa convertirá en primer lugar la imagen digital en escala de grises a una imagen en blanco y negro, teniendo previamente en cuenta todos los factores de escala con los que puede estar impreso el código en la imagen:

* Escala natural (factor de escala ***1x***). En este caso ***1*** módulo se corresponde con ***1*** píxel.
* Código ampliado (factor de escala ***2x***, ***3x***, ***4x***,...). En este caso ***1*** módulo se corresponde con un cuadrado de ***2x2***, ***3x3***, ***4x4***,... píxeles, respectivamente, siendo su valor de intensidad de gris la media aritmética de los valores de los píxeles correspondientes. El programa deberá escalar la imagen (considerando como punto fijo la esquina superior izquierda) y modificar sus dimensiones, que pasarán a ser de ***32x32***, ***21x21***, ***16x16***,... respectivamente.

A continuación, localizará matrices candidatas de ***11x11*** módulos comprobando el margen de ***2*** módulos de color blanco que la rodea. Para cada una de estas matrices candidatas, comprobará:

* El patrón de detección de posición situado en la esquina superior izquierda (cuadro negro de ***3x3*** módulos, rodeado por un margen blanco encerrado en un margen negro, ambos de ***1*** módulo),
* El separador de ***1*** módulo de color blanco, situado a la derecha y debajo del patrón de detección de posición.
* El patrón temporal formado por ***3*** módulos (negro-blanco-negro) situado tanto a la derecha de la primera fila como en la parte inferior de la primera columna.

Además, el programa deberá tener en cuenta el posible ángulo de giro del ***micro código QR*** dentro de la imagen digital, por lo que éste solo puede ser detectado cuando este ángulo sea de ***0º***, ***90º***, ***180º*** ó ***270º***.

Ejemplos:

|  |  |
| --- | --- |
| **Imagen digital en B/N** | **Micro código QR M1 detectado tras aplicar escalado y/o rotación** |
|  | Factor de escala: 1x  Localización matriz candidata:    Ángulo de giro del código: 0º |
|  | Factor de escala: 2x    Localización matriz candidata:    Ángulo de giro del código: 90º |
|  | Factor de escala: 4x    Localización matriz candidata:    Ángulo de giro del código: 180º |

|  |  |
| --- | --- |
| **Estrategia de resolución del problema** | |
| Análisis  (Pre-diseño) | Leer enunciado detenidamente e identificar (mediante análisis gramatical de la narrativa del enunciado):   * Información a procesar (sustantivos) 🡪 separar los elementos de E y de S, asignarles un identificador, un tipo de datos y posibles restricciones en sus valores. * Tareas de manipulación de información (verbos, frases verbales) 🡪 Lista de tareas 🡪 descomponer las tareas complejas en tareas más simples ampliando sus descripciones incorporando detalles. |
| Diseño 🡪 2 pasos | 1) Diseño preliminar o arquitectónico   * Diseño de datos: nuevas tipologías * Estructura del programa 🡪 diagrama de módulos   1 módulo por cada tarea diferente  Establece relaciones de dependencia (no de control)   * Interfaces entre módulos: para cada módulo del diagrama:   Nombre o identificador  Breve descripción de su actividad funcional  Lista de parámetros formales  2) Diseño detallado o procedimental:  módulo 🡪 sub-algoritmo |

**ANÁLISIS/PRE-DISEÑO**

Información:

E: Imagen en escala de grises: im (64x64 reales: [0.0,1.0]) 🡪 Tabla de valores organizados por filas

y columnas

S: micro-código QR versión M1: qr (11x11 enteros: [0,1], 1: blanco, 0: negro) 🡪 Tabla de valores

organizados por filas y columnas

Mensaje de error: ningún micro-código QR detectado

Tareas de manipulación de la información:

* Leer imagen digital de archivo de texto.
* Detectar micro-códigos QR:
  + Convertir imagen digital en escala de grises a imágenes en blanco y negro teniendo en cuenta todos los factores de escala:
    - Escalar imagen 🡪 Factores de escala: 1,2,3,4,.. Punto fijo: (0,0)
      * Calcular valor promedio de píxel escalado 🡪 tabla de valores enteros con 64/FE filas y 64/FE columnas
  + Localizar matriz 11x11 píxeles candidata
    - Comprobar margen de 2 módulos en blanco que rodea el código QR.
  + Comprobar que la matriz candidata seleccionada es un código QR:
    - Comprobar patrón de detección de posición (esquina superior izquierda).
    - Comprobar separador de 1 módulo en blanco.
    - Comprobar patrón temporal (derecha primera fila y debajo primera columna)
  + Girar matriz candidata 90º, 180º y 270º y hacer las mismas comprobaciones.
  + Imprimir en pantalla códigos QR detectados.

**DISEÑO PRELIMINAR**

**DISEÑO DE DATOS**

Objetos complejos de información identificados en el análisis del problema:

* Imagen digital escala de grises 🡪 64x64 datos reales en intervalo [0.0,1.0] organizados en filas y columnas 🡪 “array” bidimensional de 64 filas y 64 columnas de datos reales.
* Imágenes digitales en blanco y negro:
  + Escala natural (FE=1): 64x64 datos enteros en intervalo [0,1].
  + Códigos ampliados (FE=2,3,4,5,…): 32x32, 21x21, 16x16, 12x12,… datos enteros en intervalo [0,1]. Nótese que como mínimo se requieren 15x15 píxeles para representar un micro-código QR con su margen de 2 módulos, por lo que el factor de escala solo podrá llegar hasta 4.

Se representarán en las primeras filas y columnas de un “array” bidimensional de 64 filas y 64 columnas de datos enteros.

* Micro-código QR versión M1 🡪 11x11 datos enteros en intervalo [0,1] organizados en filas y columnas 🡪 “array” bidimensional de 11 filas y 11 columnas de datos enteros.

Para la identificación de los píxeles en los respectivos “arrays” bidimensionales se utilizarán dos índices cuyo significado es:

* Primer índice: representa la fila, empezando a contar por la fila superior.
* Segundo índice: representa la columna, numerándose éstas de izquierda a derecha.

|  |  |
| --- | --- |
| **Pseudo-código** | **Sintaxis de C** |
| CONST  MAXI=64  MAXQR=11  ARCHIVO="escanner001.txt" | #define MAXI 64  #define MAXQR 11  #define ARCHIVO "escaner001.txt" |
| TIPOS  tipo\_imagen: vector [MAXI, MAXI] de real  tipo\_bn: vector [MAXI, MAXI] de entero  tipo\_qr: vector [MAXQR, MAXQR] de entero  tipo\_archivo\_texto: archivo\_s de carácter | typedef float tipo\_imagen[MAXI][MAXI];  typedef int tipo\_bn[MAXI][MAXI];  typedef int tipo\_qr[MAXQR][MAXQR]; |

**ESTRUCTURA DEL PROGRAMA**

Módulo principal

localizar qr

escalar imagen

escanear imagen

comprobar marco

promediar

detectar patrones

imprimir qr

girar qr 90

extraer qr

**INTERFACES ENTRE MÓDULOS**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Pseudo-código** | | | | **Sintaxis de C** |
| Nombre módulo | Tipo parám | Nombre parám. | Tipo de datos | /\* Prototipos de funciones \*/ |
| Módulo principal |  |  |  | void escanear\_imagen(tipo\_imagen im);  void escalar\_imagen(tipo\_imagen im,int fe,tipo\_bn imbn);  int promediar(tipo\_imagen im,int i0,int j0,int fe);  int localizar\_qr(tipo\_bn im, int n);  int comprobar\_marco(tipo\_bn im,int i,int j);  void extaer\_qr(tipo\_bn im,int i0,int j0,tipo\_qr qr);  int detectar\_patrones(tipo\_qr qr);  void girar\_gr\_90(tipo\_qr qr);  void imprimir\_qr(tipo\_qr qr); |
| escanear\_imagen | S | im | tipo\_imagen |
| escalar\_imagen | E | im | tipo\_imagen |
| fe | entero |
| S | imbn | tipo\_bn |
| promediar | E | im | tipo\_imagen |
| i0 | entero |
| j0 | entero |
| fe | entero |
| S |  | entero |
| localizar\_qr | E | im | tipo\_bn |
| n | entero |
| S |  | entero |
| comprobar\_marco | E | im | tipo\_bn |
| i | entero |
| j | entero |
| S |  | lógico |
| extaer\_qr | E | im | tipo\_bn |
| i0 | entero |
| j0 | entero |
| S | qr | tipo\_qr |
| detectar\_patrones | E | qr | tipo\_qr |
| S |  | lógico |
| girar\_gr\_90 | E/S | qr | tipo\_qr |
| imprimir\_qr | E | qr | tipo\_qr |

**DISEÑO DETALLADO**

**MÓDULO PRINCIPAL**

* Declarar variables para representar la imagen original en escala de grises, la imagen digital en blanco y negro escalada, el factor de escala y el tamaño de la imagen escalada.
* Una variable contadora permitirá contabilizar los códigos QR detectados en las imágenes escaladas.
* Escanear la imagen original (leerla de un archivo de texto).
* Para cada factor de escala (1,2,3,…) construir la imagen escalada correspondiente y buscar códigos QR en la misma 🡪 repetición: una iteración para cada factor de escala. Diseño del bucle:

|  |  |  |  |
| --- | --- | --- | --- |
| Factor de escala  (fe) | Tamaño imagen escalada  (n) | n≥MAXQR+4? | fe🡨1  n🡨MAXI |
| 1 | n🡨MAXI | V: escalar\_imagen(im,fe,imbn)  localizar\_qr(imbn,n) | V: escalar\_imagen(im,fe,imbn)  localizar\_qr(imbn,n)  fe🡨fe+1  n🡨MAXI/fe |
| 2 | n🡨MAXI/2 | V: escalar\_imagen(im,fe,imbn)  localizar\_qr(imbn,n) | V: escalar\_imagen(im,fe,imbn)  localizar\_qr(imbn,n)  fe🡨fe+1  n🡨MAXI/fe |
| 3 | n🡨MAXI/3 | V: escalar\_imagen(im,fe,imbn)  localizar\_qr(imbn,n) | V: escalar\_imagen(im,fe,imbn)  localizar\_qr(imbn,n)  fe🡨fe+1  n🡨MAXI/fe |
| ... | ... | ... | ... |
| i | n🡨MAXI/i | F: | F: |

* + Condición de terminación: que el tamaño de la imagen escalada (n) sea inferior al mínimo requerido para contener un micro-código QR (MAXQR+4).

|  |  |
| --- | --- |
| **Pseudo-código** | **Sintaxis de C** |
| Algoritmo microCodigoQR  Var im: tipo\_imagen  imbn: tipo\_bn  fe: entero { Factor de escala }  n: entero { Tamaño imagen digital en BN }  cont: entero { cont. códigos QR detectados }  Inicio  escribir("CODIGOS MICRO QR")  escanear\_imagen(im);  fe🡨1  cont🡨0  n🡨MAXI  Mientras(n≥MAXQR+4) Hacer  Escribir("Factor de escala: ",fe)  escalar\_imagen(im,fe,imbn)  cont🡨cont+localizar\_qr(imbn,n)  fe🡨fe+1  n🡨MAXI/fe  Fin\_Mientras  Si(cont=0) Entonces  Escribir("Ninguna imagen qr detectada")  Fin\_si  Fin\_Algoritmo\_Principal | int main(){  tipo\_imagen im;  tipo\_bn imbn;  int fe; /\* Factor de escala \*/  int n; /\* Tamaño de la imagen digital en BN \*/  int cont;/\* contador de códigos QR detectados \*/  system("cls");  printf("CODIGOS MICRO QR\n");  printf("================\n\n");  escanear\_imagen(im);  fe=1;  cont=0;  n=MAXI;  while(n>=MAXQR+4){  printf("Factor de escala: %d\n",fe);  escalar\_imagen(im,fe,imbn);  cont+=localizar\_qr(imbn,n);  fe++;  n=MAXI/fe;  }  if(cont==0){  printf("\nNinguna imagen qr detectada\n");  }  getch();  return 0;  } |

**escanear\_imagen**

* Leer imagen de archivo de texto y almacenarla en un "array" bidimensional.
* Abrir archivo de texto con la imagen, e inicializar índices de fila y columna del “array” bidimensional a 1.
* Recorrido secuencial del archivo 🡪 lectura anticipada del primer valor real del archivo, comprobación de que no es el final del archivo, procesamiento del dato leído y lectura actualizada del siguiente valor real del archivo.
  + Procesamiento del dato leído 🡪 se almacena en la posición [i,j] del “array”, se incrementa el nº de columna j en una unidad, y si el nuevo valor supera el nº de columnas (j>MAXI), se reinicia a 1 y se incrementa el nº de fila en una unidad.
* Por seguridad, se ha añadido una segunda condición de salida (i>MAXI) al recorrido secuencial del archivo para evitar que almacenemos los datos leídos del archivo fuera de los límites declarados del “array”.

|  |  |
| --- | --- |
| **Pseudo-código** | **Sintaxis de C** |
| Procedimiento escanear\_imagen(im: tipo\_imagen (S))  Var f: tipo\_archivo\_texto  i,j: entero  x: real  Inicio Abrir(f,L,ARCHIVO)  i🡨1  j🡨1  Leer\_a(f,x)  Mientras((i≤MAXI)y(no fdaf(f)))  Hacer im[i][j]🡨x  j🡨j+1  Si(j>MAXI)  Entonces j🡨1  i🡨i+1  Fin\_si  Leer(f,x)  Fin\_Mientras  Cerrar(f)  Fin\_Procedimiento | void escanear\_imagen(tipo\_imagen im){  FILE \*f;  int i,j;  float x;  if((f=fopen(ARCHIVO,"r"))==NULL){  printf("\nError: archivo no encontrado: %s",ARCHIVO);  getch();  exit(0);  }else{  i=0;  j=0;  fscanf(f," %f",&x);  while((i<MAXI)&&(!feof(f))){  im[i][j]=x;  j++;  if(j==MAXI){  j=0;  ++i;  }  fscanf(f," %f",&x);  }  fclose(f);  }  } |

**escalar\_imagen**

* Calcular dimensiones de la imagen escalada: n🡨MAXI DIV fe
* Calcular valor promedio de cada píxel escalado: recorrido secuencial del “array” bidimensional de la imagen escalada 🡪 repetición doble controlada por los contadores de iteración **i** y **j** vinculados a los índices que representan las filas y columnas 🡪 en cada iteración del bucle interno se calcula el valor del píxel escalado **(i,j)** promediando un cuadrado de píxeles de la imagen digital original de lado **fe**, que empiezan en la posición (esquina superior izquierda): **(1+(i-1)\*fe,1+(j-1)\*fe)**. Ejemplo (fe=3):

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |
| 2 | 4 |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |
|  | | 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | | | 2 | | |

* Nótese que en lenguaje de programación C, la indexación empieza por **0**, por lo que la posición inicial del cuadro de píxel correspondiente al píxel escalado **(i,j)** se simplifica a **(i\*fe,j\*fe)**. Ejemplo (fe=3):

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |
| 1 | 3 |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |
|  | | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | | | 1 | | |

|  |  |
| --- | --- |
| **Pseudo-código** | **Sintaxis de C** |
| Procedimiento escalar\_imagen(im: tipo\_imagen (E),  fe: entero (E), imbn: tipo\_bn (S))  Var i,j: entero  n: entero  Inicio n🡨MAXI/fe  Desde i=1 Hasta n  Desde j=1 Hasta n  Hacer imbn[i][j]🡨promediar(im,  1+(i-1)\*fe,1+(j-1)\*fe,fe)  Fin\_desde  Fin\_desde  Fin\_Procedimiento | void escalar\_imagen(tipo\_imagen im,int fe,  tipo\_bn imbn){  int i,j;  int n;  n=MAXI/fe;  for(i=0;i<n;++i)  for(j=0;j<n;++j)  imbn[i][j]=promediar(im,  i\*fe,j\*fe,fe);  } |

**promediar**

* Recorrido secuencial parcial del sub-array bidimensional que empieza en la posición (i0,j0) y acaba en la posición final (i0+fe-1,j0+fe-1) 🡪 repetición doble controlada por contadores de iteraciones asociados a los índices que representan las filas y las columnas.
* Usar un acumulador de sumas y dividir finalmente por el nº total de píxeles a promediar.
* Determinar mediante una selectiva si el píxel promediado es blanco o negro.

|  |  |
| --- | --- |
| **Pseudo-código** | **Sintaxis de C** |
| Función promediar(im: tipo\_imagen im (E),  i0: entero (E), j0: entero (E),  fe: entero (E)): entero  Var x: real  i,j: entero  Inicio x🡨0  Desde i=i0 Hasta (i0+fe-1)  Hacer Desde j=j0 Hasta (j0+fe-1)  Hacer x🡨x+im[i,j]  Fin\_desde  Fin\_desde  x🡨x/(fe\*fe)  Si (x<0.5)  Entonces Devolver 0  Sino Devolver 1  Fin\_si  Fin\_función | int promediar(tipo\_imagen im,int i0,int j0,int fe){  float x;  int i,j;  x=0;  for(i=i0;i<i0+fe;++i)  for(j=j0;j<j0+fe;++j)  x+=im[i][j];  x=x/(fe\*fe);  if(x<0.5)  return 0;  else return 1;  } |

**localizar\_qr**

* Recorrido secuencial del “array” 2D de la imagen digital en blanco y negro 🡪 repetición doble controlada por dos contadores de iteración i y j asociados a los índices que representan las filas y columnas 🡪 en cada iteración:
  + Comprobar si el sub-array 2D que empieza en la posición actual de 15x15 píxeles (MAXQR+4 x MAXQR+4 píxeles) tiene un marco rectangular blanco de 2 píxeles de grosor.
    - En caso afirmativo: extraer un sub-array de 11x11 píxeles (MAXQRxMAXQR píxeles) a partir de la posición (i+2,j+2), y comprobar si contiene el patrón de detección en el mismo, o bien al girarlo 90º, 180º y 270º lo contiene. Nota: para girar el "array" 2D 180º, se gira 90º y después se vuelve a girar otros 90º.
    - En caso negativo: pasar a la siguiente posición del "array".
* Nótese que los recorridos secuenciales no son completos 🡪 no llegan a la última fila/columna respectivamente ya que el marco rectangular que hay que detectar tiene 15 píxeles de lado:

|  |
| --- |
| **Pseudo-código** |
| Función localizar\_qr(im: tipo\_bn (E),  n: entero (E)): entero  Var enc: entero  qr: tipo\_qr  i,j,k: entero  Inicio enc🡨0  i🡨1  Mientras(i≤n-MAXQR-4)  Hacer j🡨1  Mientras(j≤n-MAXQR-4)  Hacer Si(comprobar\_marco(im,i,j))  Entonces extaer\_qr(im,i+2,j+2,qr)  Escribir("Posible patron")  Escribir("Girando patron 0 grados")  Si(detectar\_patrones(qr))  Entonces Escribir("Patron detectado:")  enc🡨1  imprimir\_qr(qr)  Fin\_i  Desde k=1 Hasta 3  Hacer Escribir("Girando patron ",k\*90," grados")  girar\_gr\_90(qr)  Si(detectar\_patrones(qr))  Entonces Escribir("Patron detectado:")  enc🡨1  imprimir\_qr(qr)  Fin\_si  Fin\_desde  Fin\_si  j🡨j+1  Fin\_mientras  i🡨i+1  Fin\_mientras  Devolver enc  Fin\_función |
| **Sintaxis de C** |
| int localizar\_qr(tipo\_bn im, int n){  int enc;  tipo\_qr qr;  int i,j,k;  enc=0;  i=0;  while(i<n-MAXQR-4){  j=0;  while(j<n-MAXQR-4){  if(comprobar\_marco(im,i,j)){  extaer\_qr(im, i+2, j+2, qr);  printf("Posible patron\n");  printf("Girando patron %d grados\n",0);  if(detectar\_patrones(qr)){  printf("\tPatron detectado:");  enc=1;  imprimir\_qr(qr);  }  for(k=1;k<=3;++k){  printf("Girando patron %d grados\n",k\*90);  girar\_gr\_90(qr);  if(detectar\_patrones(qr)){  printf("\tPatron detectado:");  enc=1;  imprimir\_qr(qr);  }  }  }  ++j;  }  ++i;  }  return enc;  } |

**comprobar\_marco**

* Comprobar si hay un marco blanco cuadrado de 2 píxeles de grosor y de 15 píxeles de ancho (MAXQR+4) que empieza en la esquina superior izquierda de la imagen cuya posición es (i,j):
  + Recorridos secuenciales de las filas i, i+1, i+13 y i+14 (desde las columnas j a j+14) para detectar si hay un píxel negro en las mismas 🡪 repetición controlada por contador de iteraciones k asociado al nº de columna.
  + Recorridos secuenciales de las columnas j, j+1, j+13 y j+14 (desde las filas i+2 a i+12) para detectar si hay un píxel negro en las mismas 🡪 repetición controlada por contador de iteraciones k asociado al nº de fila.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| i+1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| i+2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| i+3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| i+4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| i+5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| i+6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| i+7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| i+8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| i+9 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| i+10 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| i+11 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| i+12 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| i+13 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| i+14 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | j | j+1 | j+2 | j+3 | j+4 | j+5 | j+6 | j+7 | j+8 | j+9 | j+10 | j+11 | j+12 | j+13 | j+14 |

|  |  |
| --- | --- |
| **Pseudo-código** | **Sintaxis de C** |
| Función comprobar\_marco(im: tipo\_bn (E),  i: entero (E), j: entero (E)): lógico  Var res: lógico  k: entero  Inicio res🡨verdadero  k🡨j  Mientras(res y (k<j+MAXQR+4))  Hacer Si(im[i][k]=0)ó(im[i+1][k]=0)ó  (im[i+13][k]=0)ó(im[i+14][k]=0)  Entonces res🡨falso  Sino k🡨k+1  Fin\_si  Fin\_mientras  k🡨i+2  Mientras(res y (k<i+MAXQR+2))  Hacer Si(im[k][j]=0)ó(im[k][j+1]=0)ó  (im[k][j+MAXQR+2]=0)ó  (im[k][j+MAXQR+3]=0)  Entonces res🡨falso  Sino k🡨k+1  Fin\_si  Fin\_mientras  Devolver res  Fin\_Función | int comprobar\_marco(tipo\_bn im,int i,int j){  int res;  int k;  res=1;  k=j;  while(res&&(k<j+MAXQR+4)){  if(im[i][k]==0||im[i+1][k]==0||  im[i+13][k]==0||im[i+14][k]==0) res=0;  else ++k;  }  k=i+2;  while(res&&(k<i+MAXQR+2)){  if(im[k][j]==0||im[k][j+1]==0||  im[k][j+MAXQR+2]==0||  im[k][j+MAXQR+3]==0)  res=0;  else ++k;  }  return res;  } |

**extaer\_qr**

* Extraer un "sub-array" 2D de 11x11 píxeles (MAXQRxMAXQR) de un "array" 2D a partir de la posición inicial (i0,j0) 🡪 recorrido secuencial parcial 🡪repetición doble controlada por dos contadores de iteración asociados a las filas y columnas del "sub-array" destino 🡪en cada iteración del bucle interno el elemento a almacenar en la posición actual del "sub-array" destino es el equivalente del "array" fuente desplazado i0-1 filas y j0-1 columnas

|  |  |
| --- | --- |
| **Pseudo-código** | **Sintaxis de C** |
| Procedimiento extaer\_qr(im: tipo\_bn (E), i0: entero (E),  j0: entero (E), qr: tipo\_qr (S))  Var i,j: entero  Desde i=1 Hasta MAXQR  Hacer Desde j=1 Hasta MAXQR  Hacer qr[i,j]🡨im[i+i0-1,j+j0-1]  Fin\_desde  Fin\_desde  Fin\_procedimiento | void extaer\_qr(tipo\_bn im,int i0,int j0,tipo\_qr qr){  int i,j;  for(i=0;i<MAXQR;++i)  for(j=0;j<MAXQR;++j)  qr[i][j]=im[i+i0][j+j0];  } |

**detectar\_patrones**

* Comprobaciones a efectuar para identificar el micro-código QR válido:
  + El patrón de detección de posición situado en la esquina superior izquierda (cuadro negro de ***3x3*** módulos, rodeado por un margen blanco encerrado en un margen negro, ambos de ***1*** módulo). Desglosarlo en varios recorridos secuenciales (simples y dobles) con una segunda condición de salida para el caso de que no se cumplan los requisitos del código a detectar:
    - Primera/última fila de píxeles negros.
    - Primera/última columna de píxeles negros
    - Segunda/penúltima fila de píxeles blancos.
    - Segunda/penúltima columna de píxeles blancos.
    - Cuadro interior de píxeles negros.
  + El separador de ***1*** módulo de color blanco, situado a la derecha y debajo del patrón de detección de posición 🡪 recorridos secuenciales
  + El patrón temporal formado por ***3*** módulos (negro-blanco-negro) situado tanto a la derecha de la primera fila como en la parte inferior de la primera columna 🡪 acceso directo a las posiciones correspondientes.

|  |  |
| --- | --- |
| **Pseudo-código** | **Sintaxis de C** |
| Función detectar\_patrones(qr: tipo\_qr (E)): lógico  Var enc: lógico  i,j: entero  Inico enc🡨verdadero  { patron de deteccion de posicion }  i🡨1  Mientras((i≤7)y(enc))  Hacer Si(qr[i][1]=1)ó(qr[i][7]=1)  Entonces enc🡨falso  Sino i🡨i+1  Fin\_si  Fin\_mientras  j🡨2  Mientras((j≤6)y(enc))  Hacer Si(qr[1][j]=1)ó(qr[7][j])  Entonces enc🡨falso  Sino j🡨j+1  Fin\_si  Fin\_mientras  i🡨2  Mientras((i≤6)y(enc))  Hacer Si(qr[i][2]=0)ó(qr[i][6]=0)  Entonces enc🡨falso  Sino i🡨i+1  Fin\_si  Fin\_mientras  j🡨3  Mientras((j≤5)y(enc))  Hacer Si(qr[2][j]=0)ó(qr[6][j]=0)  Entonces enc🡨falso  Sino j🡨j+1  Fin\_si  Fin\_mientras  i🡨3  Mientras((i≤5)y(enc))  Hacer j🡨3  Mientras((j≤5)y(enc))  Hacer Si(qr[i][j]=1)  Entonces enc🡨falso  Sino j🡨j+1  Fin\_si  Fin\_mientras  Si(enc)  Entonces i🡨i+1  Fin\_si  Fin\_mientras  { Separador }  i🡨1  Mientras((i≤8)y(enc))  Hacer Si(qr[i][8]=0)  Entonces enc🡨falso  Sino i🡨i+1  Fin\_si  Fin\_mientras  j🡨1  Mientras((j≤7)y(enc))  Hacer Si(qr[8][j]=0)  Entonces enc🡨0  Sino j🡨j+1  Fin\_si  Fin\_mientras  { Patron temporal }  Si(enc)  Entonces Si((qr[1][9]=1)ó(qr[1][10]=0)ó  (qr[1][11]=1)ó(qr[9][1]=1)ó  (qr[10][1]=0)ó(qr[11][1]=1))  Entonces enc🡨0  Fin\_si  Devolver enc  Fin\_Función | int detectar\_patrones(tipo\_qr qr){  int enc;  int i,j;  enc=1;  /\* patron de deteccion de posicion \*/  i=0;  while((i<7)&&(enc)){  if(qr[i][0]||qr[i][6])  enc=0;  else ++i;  }  j=1;  while((j<6)&&enc){  if(qr[0][j]||qr[6][j])  enc=0;  else ++j;  }  i=1;  while((i<6)&&(enc)){  if(qr[i][1]==0||qr[i][5]==0)  enc=0;  else ++i;  }  j=2;  while((j<5)&&enc){  if(qr[1][j]==0||qr[5][j]==0)  enc=0;  else ++j;  }  i=2;  while((i<5)&&enc){  j=2;  while((j<5)&&enc){  if(qr[i][j])  enc=0;  else ++j;  }  if(enc) ++i;  }  /\* Separador \*/  i=0;  while((i<8)&&enc){  if(qr[i][7]==0)  enc=0;  else ++i;  }  j=0;  while((j<7)&&enc){  if(qr[7][j]==0)  enc=0;  else ++j;  }  /\* Patron temporal \*/  if(enc)  if((qr[0][8]==1)||(qr[0][9]==0)||  (qr[0][10]==1)||(qr[8][0]==1)||  (qr[9][0]==0)||(qr[10][0]==1))  enc=0;  return enc;  } |

**girar\_gr\_90**

* Recorrido secuencial del “array” bidimensional para calcular cada elemento del “array” girado 🡪 repetición doble controlada por contadores de iteraciones i (filas) y j (columnas)
* Planteemos un ejemplo para analizar las operaciones a realizar cuando giramos un “array” bidimensional 90º en sentido anti-horario con pivote en la esquina inferior izquierda:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | a | b | c | d | e | f |
| 2 | g | h | i | j | k | l |
| 3 | m | n | o | p | q | r |
| 4 | s | t | u | v | w | x |
| 5 | y | z | 0 | 1 | 2 | 3 |
| 6 | 4 | 5 | 6 | 7 | 8 | 9 |
|  | 1 | 2 | 3 | 4 | 5 | 6 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | f | l | r | x | 3 | 9 |
| 2 | e | k | q | w | 2 | 8 |
| 3 | d | j | p | v | 1 | 7 |
| 4 | c | i | o | u | 0 | 6 |
| 5 | b | h | n | t | z | 5 |
| 6 | a | g | m | s | y | 4 |
|  | 1 | 2 | 3 | 4 | 5 | 6 |

* El nº de fila pasa a ser directamente el nº de columna, y el nº de columna pasa a ser el de fila pero empezando a contar por la fila de abajo: (i,j) 🡪 (MAXQR+1-j,i).
* Utilizar un “array” auxiliar para realizar la operación de giro (ya que al sobre-escribirla se perderían algunos valores originales necesarios para calcular otros valores del “array” girado), y al finalizarla se copia al “array” original.

|  |  |
| --- | --- |
| **Pseudo-código** | **Sintaxis de C** |
| Procedimiento girar\_gr\_90(qr: tipo\_qr (E/S))  Var i,j: entero  qr2: tipo\_qr  Inicio { girar 90 grados }  Desde i=1 Hasta MAXQR  Hacer Desde j=1 Hasta MAXQR  Hacer qr2[MAXQR+1-j,i]🡨qr[i,j]  Fin\_desde  Fin\_desde  { copiar matriz }  Desde i=1 Hasta MAXQR  Hacer Desde j=1 Hasta MAXQR  Hacer qr[i,j]🡨qr2[i,j]  Fin\_desde  Fin\_desde  Fin\_procedimiento | void girar\_gr\_90(tipo\_qr qr){  int i,j;  tipo\_qr qr2;  /\* girar 90 grados \*/  for(i=0;i<MAXQR;++i)  for(j=0;j<MAXQR;++j)  qr2[MAXQR-1-j][i]=qr[i][j];  /\* copiar matriz \*/  for(i=0;i<MAXQR;++i)  for(j=0;j<MAXQR;++j)  qr[i][j]=qr2[i][j];  } |

**imprimir\_qr**

* Recorrido secuencial del “array” bidimensional para escribirlo en pantalla 🡪 repetición doble controlada por contadores de iteraciones i (filas) y j (columnas):

|  |  |
| --- | --- |
| **Pseudo-código** | **Sintaxis de C** |
| Procedimiento imprimir\_qr(qr: tipo\_qr (E))  Var i,j: entero  Inicio Desde i=1 Hasta MAXQR  Hacer Desde j=1 Hasta MAXQR  Hacer Si(qr[i][j]=1)  Entonces Escribir("1")  Sino Escribir("0")  Fin\_si  Fin\_desde  Fin\_desde  Fin\_Procedimiento | void imprimir\_qr(tipo\_qr qr){  int i,j;  printf("\n\n");  for(i=0;i<MAXQR;++i){  for(j=0;j<MAXQR;++j){  if(qr[i][j])  printf("%c%c",178,178);  else printf("%c%c",176,176);  }  printf("\n");  }  } |

**CRITERIOS UTILIZADOS EN LA EVALUACIÓN DEL EXAMEN**

A modo orientativo, para la evaluación de este examen de convocatoria se desglosó la calificación en base a los sub-problemas principales a resolver descritos en la siguiente tabla. Cualquier alternativa de solución creativa e innovadora para algún aspecto del problema planteada por el alumno, fue tenida en cuenta positivamente en la evaluación. Aspectos desfavorables en la evaluación: mal dominio de la sintaxis de C (errores sintácticos destacables y continuos), diseño y utilización incorrecta de estructuras de control, de estructuras de datos y de interfaces de módulos.

|  |  |
| --- | --- |
| **Elementos a evaluar** | **Puntos** |
| Diseño preliminar: diseño de datos (nuevas tipologías) + estructura programa (diagrama de módulos) + interfaces entre módulos (prototipos de funciones) + Módulo principal | 2 |
| Leer imagen del archivo de texto | 1 |
| Escalar imagen + promediar valores de píxeles+ Convertir escala de grises a Blanco/Negro | 2 |
| Localizar código QR candidato 🡪 detectar margen | 1 |
| Detectar patrón posición + separador + patrón temporal | 2 |
| Girar QR 90º | 1 |
| Imprimir QR | 1 |
| Total | 10 |

La calificación final en la asignatura consta de tres apartados cuya contribución es la siguiente:

* Examen final: 80%
* Trabajo en Equipo (Grupo de Trabajo): 10%
* Trabajo Individual: 10%

|  |
| --- |
| **Implementación completa del programa en lenguaje de programación C** |
| #include <stdlib.h>  #include <stdio.h>  #include <conio.h>  #include <ctype.h>  #include <math.h>  #define MAXI 64  #define MAXQR 11  #define ARCHIVO "escaner001.txt"  /\* Diseño Preliminar \*/  /\* Diseño de Datos \*/  /\* Nuevos tipos de datos \*/  typedef float tipo\_imagen[MAXI][MAXI];  typedef int tipo\_bn[MAXI][MAXI];  typedef int tipo\_qr[MAXQR][MAXQR];  /\* Interfaces entre módulos \*/  /\* Prototipos de funciones \*/  void escanear\_imagen(tipo\_imagen im);  void escalar\_imagen(tipo\_imagen im,int fe,tipo\_bn imbn);  int promediar(tipo\_imagen im,int i0,int j0,int fe);  int localizar\_qr(tipo\_bn im, int n);  int comprobar\_marco(tipo\_bn im,int i,int j);  void extaer\_qr(tipo\_bn im,int i0,int j0,tipo\_qr qr);  int detectar\_patrones(tipo\_qr qr);  void girar\_gr\_90(tipo\_qr qr);  void imprimir\_qr(tipo\_qr qr);  /\* Diseño Detallado \*/  /\* Definiciones de funciones \*/  int main(){  tipo\_imagen im;  tipo\_bn imbn;  int fe; /\* Factor de escala \*/  int n; /\* Tamaño de la imagen digital en BN \*/  int cont;/\* contador de códigos QR detectados \*/  system("cls");  printf("CODIGOS MICRO QR\n");  printf("================\n\n");  escanear\_imagen(im);  fe=1;  cont=0;  n=MAXI;  while(n>=MAXQR+4){  printf("Factor de escala: %d\n",fe);  escalar\_imagen(im,fe,imbn);  cont+=localizar\_qr(imbn,n);  fe++;  n=MAXI/fe;  }  if(cont==0){  printf("\nNinguna imagen qr detectada\n");  }  getch();  return 0;  }  void escanear\_imagen(tipo\_imagen im){  FILE \*f;  int i,j;  float x;  if((f=fopen(ARCHIVO,"r"))==NULL){  printf("\nError: archivo no encontrado: %s",ARCHIVO);  getch();  exit(0);  }else{  i=0;  j=0;  fscanf(f," %f",&x);  while((i<MAXI)&&(!feof(f))){  im[i][j]=x;  j++;  if(j==MAXI){  j=0;  ++i;  }  fscanf(f," %f",&x);  }  fclose(f);  }  }  void escalar\_imagen(tipo\_imagen im,int fe,tipo\_bn imbn){  int i,j;  int n;  n=MAXI/fe;  for(i=0;i<n;++i)  for(j=0;j<n;++j)  imbn[i][j]=promediar(im,i\*fe,j\*fe,fe);  }  int promediar(tipo\_imagen im,int i0,int j0,int fe){  float x;  int i,j;  x=0;  for(i=i0;i<i0+fe;++i)  for(j=j0;j<j0+fe;++j)  x+=im[i][j];  x=x/(fe\*fe);  if(x<0.5)  return 0;  else return 1;  }  int localizar\_qr(tipo\_bn im, int n){  int enc;  tipo\_qr qr;  int i,j,k;  enc=0;  i=0;  while(i<n-MAXQR-4){  j=0;  while(j<n-MAXQR-4){  if(comprobar\_marco(im,i,j)){  extaer\_qr(im,i+2,j+2,qr);  printf("Posible patron\n");  printf("Girando patron %d grados\n",0);  if(detectar\_patrones(qr)){  printf("\tPatron detectado:");  enc=1;  imprimir\_qr(qr);  }  for(k=1;k<=3;++k){  printf("Girando patron %d grados\n",k\*90);  girar\_gr\_90(qr);  if(detectar\_patrones(qr)){  printf("\tPatron detectado:");  enc=1;  imprimir\_qr(qr);  }  }  }  ++j;  }  ++i;  }  return enc;  }  int comprobar\_marco(tipo\_bn im,int i,int j){  int res;  int k;  res=1;  k=j;  while(res&&(k<j+MAXQR+4)){  if(im[i][k]==0||im[i+1][k]==0||  im[i+13][k]==0||im[i+14][k]==0)  res=0;  else ++k;  }  k=i+2;  while(res&&(k<i+MAXQR+2)){  if(im[k][j]==0||im[k][j+1]==0||  im[k][j+MAXQR+2]==0||  im[k][j+MAXQR+3]==0)  res=0;  else ++k;  }  return res;  }  void extaer\_qr(tipo\_bn im,int i0,int j0,tipo\_qr qr){  int i,j;  for(i=0;i<MAXQR;++i)  for(j=0;j<MAXQR;++j)  qr[i][j]=im[i+i0][j+j0];  }  int detectar\_patrones(tipo\_qr qr){  int enc;  int i,j;  enc=1;  /\* patron de deteccion de posicion \*/  i=0;  while((i<7)&&(enc)){  if(qr[i][0]||qr[i][6])  enc=0;  else ++i;  }  j=1;  while((j<6)&&enc){  if(qr[0][j]||qr[6][j])  enc=0;  else ++j;  }  i=1;  while((i<6)&&(enc)){  if(qr[i][1]==0||qr[i][5]==0)  enc=0;  else ++i;  }  j=2;  while((j<5)&&enc){  if(qr[1][j]==0||qr[5][j]==0)  enc=0;  else ++j;  }  i=2;  while((i<5)&&enc){  j=2;  while((j<5)&&enc){  if(qr[i][j])  enc=0;  else ++j;  }  if(enc) ++i;  }  /\* Separador \*/  i=0;  while((i<8)&&enc){  if(qr[i][7]==0)  enc=0;  else ++i;  }  j=0;  while((j<7)&&enc){  if(qr[7][j]==0)  enc=0;  else ++j;  }  /\* Patron temporal \*/  if(enc)  if((qr[0][8]==1)||(qr[0][9]==0)||(qr[0][10]==1)||  (qr[8][0]==1)||(qr[9][0]==0)||(qr[10][0]==1))  enc=0;  return enc;  }  void girar\_gr\_90(tipo\_qr qr){  int i,j;  tipo\_qr qr2;  /\* girar 90 grados \*/  for(i=0;i<MAXQR;++i)  for(j=0;j<MAXQR;++j)  qr2[MAXQR-1-j][i]=qr[i][j];  /\* copiar matriz \*/  for(i=0;i<MAXQR;++i)  for(j=0;j<MAXQR;++j)  qr[i][j]=qr2[i][j];  }  void imprimir\_qr(tipo\_qr qr){  int i,j;  //system("cls");  printf("\n\n");  for(i=0;i<MAXQR;++i){  //printf("\t");  for(j=0;j<MAXQR;++j){  if(qr[i][j])  printf("%c%c",178,178);//printf("1");  else printf("%c%c",176,176);//printf("0");  }  printf("\n");  }  } |