ICOM2015 – Examen Final B

Diciembre 2015

Notas:

1. Al finalizar, enviar por e-mail los archivos fuente de cada ejercicio con nombre APELLIDO_NOMBRE_Ejer_N.c a icom.cabib@gmail.com
2. Uso de prácticos: **se pueden utilizar los trabajos prácticos propios realizados**.
3. Uso de Internet**: solo para la consulta de referencias de funciones de C**.

PROBLEMA 1 - *Integración numérica*

Existen gran cantidad de métodos de integración numérica que permiten calcular áreas en forma aproximada. Uno de ellos se basa en ir dividiendo el intervalo de integración en sub-intervalos y reemplazar en cada uno de esos sub-intervalos la función original por parábolas, de esta forma el área en uno de esos intervalos será:

$$\int_{x1}^{x2} f(x)dx \approx \frac{x2-x1}{6}\left[f(x1)+4\,f\left(\frac{x1+x2}{2}\right)+f(x2)\right]$$

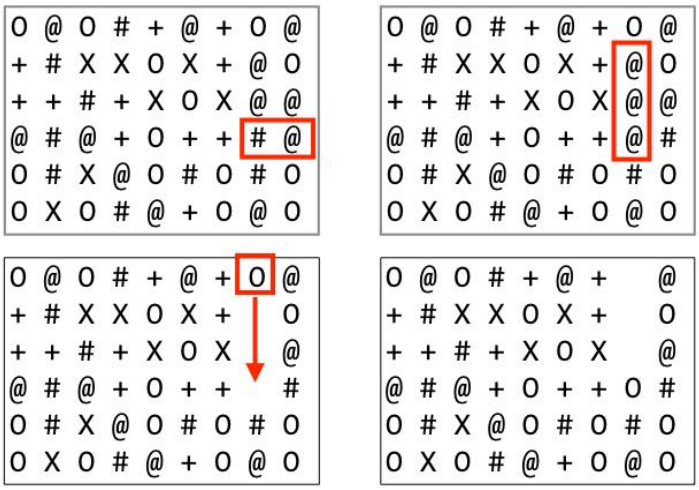
Se solicita implementar la función:

```
typedef double (*Fun_t) (double value);
double calculaArea(Fun_t f, double a, double b, double error);
```

Que calcule el área aproximada de la función f en el intervalo [**a**,**b**], utilizando este método. La función deberá ir haciendo subdivisiones (1, 2, 4, 8, etc.) hasta que la diferencia de área entre dos iteraciones sucesivas de un valor menor a **`error`**.

PROBLEMA 2 – *Ascii Crush*

El conocido juego de celulares Candy Crush se desarrolla en un tablero rectangular de WIDTH x HEIGHT casilleros, donde se colocan fichas (o candies) de NTYPES tipos distintos. Se inicia el juego colocando fichas al azar en todo el tablero. En cada turno, el jugador puede intercambiar de lugar una ficha ("swappear") con otra adyacente que se encuentre arriba, abajo, a la izquierda o a la derecha, con el objetivo de que tres o más fichas iguales queden juntas en forma vertical u horizontal. Las fichas así alineadas se eliminan del tablero. El o los huecos así generados son completados por fichas que están arriba, si las hubiere, que caen a ocupar los lugares de las fichas eliminadas. Finalmente, los casilleros vacíos generados por este procedimiento son ocupados por nuevas fichas elegidas al azar.



Por ejemplo, en la figura se muestra el intercambio de la ficha # con su contigua a la derecha, @. Este intercambio da lugar a tres fichas '@' contiguas alineadas verticalmente. Estas fichas se eliminan del tablero, y la ficha 'O' que queda arriba de la columna, 'cae'. Finalmente, los tres espacios vacíos son completados con fichas generadas al azar.

En este problema se propone programar algunas funciones necesarias para implementar Ascii Crush, donde las fichas son números enteros del 1 a NTYPES, y se asume que si un casillero contiene al

entero cero, entonces está vacío. Para ello se requiere programar las siguientes funciones

a. Una función con prototipo

`void fillBoard(int board[HEIGHT][WIDTH]);`

que recorre el tablero `board` buscando los casilleros vacíos, y los llena con algún número al azar entre 1 y NTYPES.

b. Una función con prototipo

`int swapCandy(int board[HEIGHT][WIDTH], int i, int j, int dir);`

que intercambia la ficha en la posición `(i,j)` con la adyacente en la dirección `dir` (que puede ser arriba, abajo, izquierda o derecha). Devuelve 1 si al hacer el swap 3 o más fichas quedan juntas, 0 en caso contrario. Si devuelve 0 las fichas no se tienen que intercambiar

c. Una función con prototipo

`int candyCrush(int board[HEIGHT][WIDTH]);`

que recorre el tablero `board` buscando las fichas contiguas que se repiten tres o más veces (en forma vertical u horizontal), y las elimina del tablero. Observe que es posible que una ficha puede ser común a dos alineaciones, una en vertical, y otra en horizontal. Devuelve 1 si eliminó alguna ficha y 0 en caso contrario.

d. Una función con prototipo

`void slideBoard(int board[HEIGHT][WIDTH]);`

que se llamará a continuación de candyCrush (si esta eliminó fichas). Ésta función recorre el tablero board buscando los casilleros vacíos en cada columna, y mueve las fichas que correspondan hacia abajo.

En el archivo `crush.c` encontrará los prototipos de las funciones, y la función printBoard que imprime el tablero usando algunos símbolos.

PROBLEMA 3 – *Descompresión de imágenes monocromáticas*

Una imagen monocromática es una imagen en donde solo pueden aparecer dos colores. Una manera de almacenar en memoria imágenes bidimensionales de este tipo es utilizando mapas de bits, es decir, representar los colores de los pixeles de una fila de la imagen como una secuencia de bits, interpretando que un bit en 0 (cero) representa un color y un bit en 1 el color restante.

Suponiendo, para simplificar, que se está trabajando en una plataforma en donde un entero sin signo ocupa 4 bytes (32 bits), para una imagen de n filas por m columnas, el mapa de bits de cada fila se implementan como un arreglo de (m+31)/32 enteros sin signo.

Así, el color del pixel en la posición **`X`** de la fila **`Y`**-ésima de la imagen está representado por el bit (**`X % 32`**)-ésimo del entero que está en el índice (**`X / 32`**) del vector de enteros que representa la fila **`Y`**-ésima de pixeles.

Por otro lado, existen diferentes técnicas de compresión de datos que apuntan a reducir más el volumen de información que describe a una imagen. Una de tales técnicas es la **RLE** (**R**un-**L**ength-**E**ncoding) que se basa en representar las secuencias de datos con el mismo valor consecutivo (repetido), como un único valor seguido por el número de repeticiones.

Suponga que se cuenta con una imagen comprimida, cuyo formato es:

`ancho alto nruns`
`rl`$_0$
`rl`$_1$
`rl`$_2$
`rl`$_3$

…

en donde **ancho** y **alto** definen las dimensiones de la imagen, **nruns** indica la cantidad de runs que definen la imagen, y los $\mathbf{rl_x}$ son los run-lengh codes. Estos se encuentran codificados como caracteres sin signo (8 bits sin signo) de tal manera que el bit 7 representa el color (**color = ($\mathbf{rl_x}$ >> 7)**) y el número de repeticiones de ese color está indicado en los 7 bits menos significativos (**nrep = $\mathbf{rl_x}$ & 0x7F**).

En la compresión, los runs se generan recorriendo la imagen fila por fila, de izquierda a derecha, de arriba hacia abajo.

Suponga que cuenta con los servicios (decomp.c):

```
// crea una instancia para almacenar una imagen
// monocromática de las dimensiones dadas
MonoImg_t *CreaImagen(int ancho, int alto);

// Destruye la imagen liberando recursos
void DestruyeImagen(MonoImg_t *pImg);

// Impone un pixel con un color determinado (0 o 1)
void SetPixel(MonoImg_t *pImg, int X, int Y, int color);

// Retorna el color de un pixel determinado (color 0 o 1)
int GetPixel(MonoImg_t *pImg, int X, int Y);

// Muestra la imagen en pantalla
void Imprime(MonoImg_t *pImg);
```

Se solicita realizar la función:

```
MonoImg_t *Decompress(cons char *imgName);
```

Esta función debe leer una imagen comprimida desde el archivo **imgName** dada en el formato indicado anteriormente y crear la imagen que ésta representa.

Utilizar el esqueleto de programa **decomp.c** y las funciones ya implementadas. El archivo **img.compressed** contiene una imagen comprimida de prueba.