ICOM2019 – 2do Parcial

28 de noviembre de 2019

Notas:

1. Al finalizar, enviar por e-mail los archivos fuente de cada ejercicio con nombre **APELLIDO_NOMBRE_Ejer_N.cpp** 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/C++**

Problema 1: Límite de una serie

Escribir una función que calcule el límite de una serie. La misma debe recibir el número máximo de iteraciones '**M**', el término general '**serie**' como un functor que define la forma de calcular el término i-ésimo de la serie (s_i) y una cota máxima del error '**cota_error**'. Si tras el número máximo de iteraciones la serie no converge, el programa debe indicar la falta de convergencia a través de una excepción del tipo **NotConvergeException**.

```
class Serie {
    public:
        // retorna el valor del termino i-esimo de la serie
        virtual double operator()(int i) = 0;
};

struct NotConvergeException {
};


double limiteSerie(int M, Serie &serie, double cota_error);
```

La idea es generar clases que hereden de la clase **Serie** y que redefinan la forma de calcular el i-esimo término manteniendo el estado necesario dentro de los miembros privados.
Si bien es muy mala, en la función **limiteSerie** defina la convergencia de la serie como la condición |**serie(i)**| < cota_error

Definir y probar los functores y la función solicitada para representar y calcular los límites de las siguientes series:

- $s_n = a*r^n$ Con a = 1, y r = 0.5, la serie tiende a 2
- $s_n = 1/\pi^n$ La serie tiende a $\pi/(\pi-1)$
- $s_n = (-3)^n/(2n+1)$ La serie no cenverge

Problema 2: Sistema de menús

Se desea representar/navegar un sistema de menú de opciones. Cada opción del menú está definida por una etiqueta (string que representa el título del menú), y ninguna o más subopciones que son también opciones.

Un sistema de opciones de este tipo puede ser definido a través de un archivo en donde figura cada opción y la cantidad de subopciones que contiene, las que están definidas a continuación, por ejemplo:

```
raiz 3              -> Opción raíz, tiene 3 subopciones
menu0 0             -> subopción de raíz, no tiene subopciones
menu1 3             -> subopción de raíz, tiene 3 subopciones
menu10 0            -> subopción de menu1, no tiene subopciones
menu11 0            -> subopción de menu1, no tiene subopciones
menu12 0            -> subopción de menu1, no tiene subopciones
menu2 1             -> subopción de raiz, tiene 1 subopciones
hola20 0            -> subopción de menu2, no tiene subopciones
```

Se desea que el usuario pueda seleccionar una de las opciones que a su vez contiene otro menu. De forma que siempre se está mostrando un menu en la pantalla y siempre se tiene la opción de salir al nivel superior.

En el archivo Menus.cpp se da un esqueleto del problema. Implementar los métodos faltantes (marcados con **ToDo**) y probar el sistema con el **main** dado (ese **main** no tiene fin, para terminar el proceso hay que dar **Ctrl-C**).

Problema 3: Cámara de vigilancia

La señal de video que entrega una cámara de vigilancia puede ser interpretada por la computadora como una secuencia de imágenes en tonos de grises.

Un algoritmo sencillo para detectar movimiento es el **Método de Resta de Fondo**, se basa en ir comparando las imágenes que se van tomando contra una imagen de referencia, ambas en escala de grises, y condicionando las comparaciones con un umbral de detección para filtrar el ruido.

La resta de imágenes da como resultado una nueva imagen que registra el cambio. Los pixeles de la imagen Resta que tienen un valor de intensidad menor que el umbral pasan a tener intensidad 0.

Resulta también útil realizar un seguimiento del "objeto" que ingresó a la imagen. Si solo un objeto ingresa a la imagen, se puede indicar la posición del objeto a través de su centroide en la imagen resta.

Una forma en que se puede calcular el centroide es calcular su centro de masa con respecto a X y con respecto a Y, tomando como peso la intensidad de cada pixel.

Dados los UDTs siguientes, completar los métodos faltantes (marcados con **`ToDo`**). En el archivo **`vigia.cpp`** hay un esqueleto del problema propuesto y un **`main`** que permite ejercitar la solución (ese **`main`** no tiene fin, para terminar el proceso hay que dar **`Ctrl-C`**). También están disponibles un conjunto de imágenes de prueba (**`bg.img`**, **`image1.img`**, etc.)

```
using Gris = unsigned char;

struct Punto2D {
    unsigned int x;
    unsigned int y;
};

class ImagenGris {
    public:
        ImagenGris(int ancho_, int alto_);       // ToDo
        Gris getPixel(int i, int j) const;       // ToDo
        void setPixel(int i, int j, Gris color); // ToDo
        int getAncho() const;                    // ToDo
        int getAlto() const;                     // ToDo
        Punto2D centroide() const;               // ToDo

    private:
        int ancho, alto;
        vector<Gris> pixels;
};

class CamProcessor {
    public:
        CamProcessor(const ImagenGris &back, Gris th) : background(back), threshold(th) {}

        // Solicita el procesamiento de una nueva imagen proveniente
        // de la cámara, el centroide de la imagen resta entre este nuevo frame
        // y el background será retornada como resultado del procesamiento
        Punto2D procFrame(const ImagenGris &frame);         // ToDo
    private:
        const ImagenGris background;   // imagen gris de fondo
        Gris threshold;                // umbral de detección
};
```