ICOM2019 – 1er Parcial

10 de octubre 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: Generador de oraciones

Se desea implementar un generador de frases aleatorias de una estructura determinada, el UDT **`GenFrase`**. Para cumplir con su objetivo deberá tener como miembros cuatro conjuntos de palabras: **`articulos`**, **`sustantivos`**, **`verbos`** y **`preposiciones`**, todos ellos del tipo **`vector<string>`**, que definen los elementos a ser utilizados en las frases. Estos conjuntos son cargados desde archivos utilizando el método **`cargaConjuntos`**.

La estructura de una frase está descripta por un vector de elementos de la enumeración **`TipoPalabra`** (por ejemplo: **`{ARTICULO, SUSTANTIVO, VERBO, PREPOSICION, ARTICULO, SUSTANTIVO}`**).

El método **`generaFrase`** recibe como argumento la estructura de la frase a generar. Cada vez que es activado, generará y retornará una frase donde los elementos son sacados de los conjuntos respectivos en forma aleatoria. Las palabras deben estar separadas con espacios y la frase debe comenzar con mayúscula y terminar con punto.

Se solicita:

a. Impementar el método **`cargaConjuntos`**.

b. Implementar el método **`generaFrase`**.

c. Implementar un **`main`** que instancie un objeto del tipo **`GenFrase`**, cargue sus conjuntos de palabras y genere algunas frases a acuerdo a una estructura de frase.

```
struct GenFrase {
   enum TipoPalabra { ARTICULO, SUSTANTIVO, VERBO, PREPOSICION };

   vector<string> articulos;
   vector<string> sustantivos;
   vector<string> verbos;
   vector<string> preposiciones;

   void cargaConjuntos(const string &fname_articulos,
        const string &fname_sustantivos,   const string &fname_verbos,
        const string &fname_preposiciones);

   string generaFrase(const vector<TipoPalabra> &estructura_frase);
};
```

Nota: los archivos articulos.txt, sustantivos.txt, verbos.txt y preposiciones.txt son ejemplos de archivos a cargar.

Problema 2: jeringozo

El **Jeringozo** es una forma graciosa de hablar que consiste en intercalar sílabas entre medio de una palabra en forma reiterada. En una de sus formas más simples, consiste en insertar luego de cada vocal la letra p seguida de la misma vocal

Hola, ¿como estas? => Hopolapa, ¿copomopo epestapas?

Y como caso especial, luego de la **y** se agrega **pi**, pero sólo cuando está sola

Bien, ¿y vos? => Bipiepen, ¿ypi vopos?

Se solicita implementar dos funciones:

a. Recibe un string en español y retorna su traducción al jeringozo.

```
string esp2jeringozo(const string &s);
```

b. Recibe un string en jeringozo y retorna su traducción al español.

```
string jeringozo2esp(const string &s);
```

Notas: No utilice acentos. Pero sí tenga en cuenta mayúsculas. Puede resultarle útil implementar algunas funciones auxiliares. Sugerimos probar sus funciones con las siguientes frases: "`Yo tambien estoy, ¿y vos?`","`y`", "`Y, ¡eramos tan pobres!`".

Problema 3: Islas de grafeno

Para depositar grafeno sobre una superficie metálica se utiliza la técnica CVD (Chemical Vapor Deposition) donde se inserta un sustrato metálico y se modula el flujo de gases reactivos en una cámara a alta temperatura. Experimentalmente se ha detectado que durante la deposición de grafeno sobre superficies metálicas se forman islas cuyo número y tamaño depende de las presiones parciales de los gases.

Por medio de STM (Microscopio Electrónico de Barrido por efecto túnel) se pueden obtener imágenes en escala de grises con intensidades entre 0 y 1023. Se desea implementar un software que, a partir de una imagen, determine automáticamente las islas de grafeno presentes.

El controlador del STM almacena imágenes en un canal de 10 bits sin comprimir. Esto es, una imagen en escala de grises con intensidades entre 0 y 1023.

```
struct STM_image {
       unsigned int width;
       unsigned int height;
       vector<unsigned short> pixels;

       unsigned int ij2index(int i, int j) {
           assert(i >= 0 && i < (int) width && j >= 0 && j < (int) height);
           unsigned int pixelIndex = i * width + j;
           assert(pixelIndex < pixels.size());
           return pixelIndex;
       }
       unsigned short getPixel(int i, int j) {
           unsigned int pixelIndex = ij2index(i, j);
           return pixels[pixelIndex];
       }
       void setPixel(int i, int j, unsigned short value) {
           unsigned int pixelIndex = ij2index(i, j);
           pixels[pixelIndex] = value;
       }
};
```

Dado que la densidad de estados del grafeno es sensiblemente diferente a la del metal, el controlador puede realizar un pre-procesamiento de la imagen y resaltar las islas de grafeno con mayor intensidad que el metal.

Una manera simplificada de caracterizar una isla es a través de los puntos que la forman:

```
struct Punto2D {
    int x, y;
};
struct IslaGrafeno {
       vector<Punto2D> puntos;
};
```

Se desea diseñar e implementar un **`GrafenoAnalizer`**: un tipo que posee un método **`analize`** que recibe una **`STM_image`** como argumento y termina retornando el vector de islas (**`vector<IslaGrafeno>`**) presentes en la imagen recibida.

```
struct GrafenoAnalizer {
    GrafenoAnalizer(unsigned short threshold_) {
        threshold = threshold_;
    }
    vector<IslaGrafeno> analize(STM_image img);
    // ...
    unsigned short threshold;
};
```

Para encontrar las islas se puede recorrer la imagen pixel a pixel. Si el valor de un pixel [Fila, Columna] es mayor que un nivel de disparo dado **`threshold`**, entonces se ha encontrado una isla: se procede a encontrar todos sus puntos almacenándolos en una **`IslaGrafeno`** y se continúa recorriendo el vector de pixeles.

HINT: Una vez ubicada una isla piense en utilizar un algoritmo parecido al **`FloodFill`** visto en la práctica para ir encontrando los puntos, pintándolos como para evitar que vuelva a ser encontrados y agregándolos al vector que definirá la isla.