ICOM2022 – 1er Parcial

22 de septiembre de 2022

Notas:

1. Uso de prácticos: **se pueden utilizar los trabajos prácticos propios realizados.**
2. **Respete** los prototipos y consignas
3. Entregue cada ejercicio en un archivo con el siguiente formato:
   APELLIDO_NOMBRE_EJ#.cpp (no se confunda el .cpp con el .cbp del codeblocks)

Problema 1: Anagramas (0.20)

Un **anagrama** es una palabra o frase que resulta de la transposición de las letras de otra palabra o frase. Dicho de otra forma, una palabra es anagrama de otra si las dos tienen las mismas letras, con el mismo número de apariciones, pero en un orden diferente.
Implemente una función que determine si dos strings son anagramas. Por ejemplo "**I am Lord Voldemort**" es un anagrama de "**Tom Marvolo Riddle**". Ignore cualquier caracter que no sea una letra y no distinga entre mayúsculas y minúsculas. Respete el siguiente prototipo:

```
bool isAnagram(const std::string &s1, const std::string &s2);
```

Problema 2: Impresión de enteros en formato binario (0.20)

Implemente una función **recursiva** y una **no recursiva** para imprimir un número entero positivo en formato binario. Hágalo sin utilizar ninguno de los siguientes operadores: (/, %). **Hint**: utilice operaciones de bits.
Respetar los siguientes prototipos:

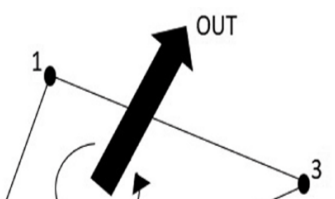
```
//version recursiva
void print_binary_rec( unsigned int n);

//version no recursiva
void print_binary( unsigned int n);
```

Problema 3: **STL** (**ST**ereo**L**ithography) (0.60)

STL Es un formato de archivo para describir modelos de objetos 3D utilizando triángulos para **teselar** la superficie del objeto. Este tipo de archivos son generados por software de diseño CAD (como **fusion360**) y se utilizan mucho en impresión 3D, como input para los **slicers** que generan otro archivo (**Gcode** por ej.) con las instrucciones que debe seguir la impresora para imprimir el objeto. Si bien suelen ser archivos binarios para optimizar el espacio que ocupan (suelen ser muy grandes), existe también una versión ASCII. El formato ASCII, como muestra la siguiente figura, es bastante simple.

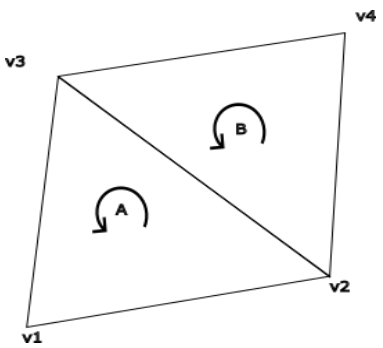
Comienza con el keyword **solid** seguido de un espacio y opcionalmente un string (sin espacios) con el *nombre* del modelo. A continuación, aparecerá un número indeterminado de triángulos que conforman la superficie del objeto. Cada triángulo comienza con el keyword **facet normal** seguido de 3 números de punto flotante separados por espacios (n_i n_j n_k), que es un vector unitario en 3D, perpendicular al plano del triángulo y apuntando hacia afuera del objeto. Luego, aparecerá el keyword **outer loop**, que contiene los 3 vértices del triángulo (**vertex**) separados por espacios, ordenados de forma que se recorre el perímetro en forma contraria a las agujas del reloj. Siguiendo la regla de la mano derecha, el pulgar apunta en la misma dirección que el vector **facet normal**.

solid *name*
(**facet normal** n_i n_j n_k / **outer loop** / **vertex** $v1_x$ $v1_y$ $v1_z$ / **vertex** $v2_x$ $v2_y$ $v2_z$ / **vertex** $v3_x$ $v3_y$ $v3_z$ / **endloop** / **endfacet**) +
endsolid *name*



El **outer loop** se cierra con el keyword **endloop**, y el triángulo (**facet normal**) se cierra con el keyword **endfacet**. Luego del último triángulo, el modelo (**solid**) se cierra con el keyword **endsolid**, que opcionalmente puede contener el nombre del modelo (aunque es muy raro).
Para que uno de estos archivos sea válido, además de seguir el formato especificado anteriormente debe cumplir con los siguientes requisitos:

1. El vector **facet normal** debe corresponder con el que se puede calcular a partir de los vértices, siguiendo la regla de la mano derecha. **Hint:** utilice el producto cruz (ver ejercicio 11 práctica 3) para calcular el facet normal a partir de los vértices.
2. Cada lado de un triángulo se comparte con <u>exactamente</u> 1 de los otros triángulos. Y en ese otro triángulo, dicho lado se recorre en el sentido contrario.
3. El objeto debe estar localizado en el octante positivo (x>=0, y>=0 y z>=0).



Aunque no es obligatorio, también se recomienda que los triángulos estén ordenados en valores crecientes de **z**, para optimizar el trabajo del slicer.
En el archivo **block100.stl** se encuentra un modelo de un cubo de 100x100x100. Y en el archivo **stl.cpp** encontrará el esqueleto de un programa para: leer un archivo stl, verificar su validez, optimizarlo para slicer, cambiar su escala y serializarlo (escribirlo a un archivo con el formato STL ASCII).