UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas

Evaluación de Aprendizaje (EdA2).

Lea atentamente este documento antes de comenzar a resolver la ejercitación planteada.

Debe resolver la siguiente problemática eligiendo la mejor estrategia.

Acerca de qué se trata el problema.

Una empresa maneja sus pedidos a través de un portal de internet. Cierto día debido a un incidente técnico el centro de cómputos colapsó, por lo que se debe recuperar la información del día a través de algunos archivos de copias de seguridad (*backup*).

Se dispone de dos archivos de texto que contienen registros correspondientes a pedidos de productos de clientes, en diferentes nodos. Estos archivos no tienen ningún ordenamiento útil ya que fueron guardados a medida que se registraban los pedidos. Puede haber uno o más de un pedido por cada cliente a lo largo del día. Cada pedido puede estar formado por uno o más registros. Cada cliente que hizo un pedido puede tener uno o más registros en cada archivo o sólo en uno.

En un primer análisis, debido a una configuración dispar de los nodos, los archivos tendrán un formato diferente. A continuación, se ve un ejemplo de su contenido (no se guíe por el hecho de que están ordenados, sólo es un ejemplo de la información almacenada):

Ejemplo del archivo [**ped_A.txt**]

```
      Tipo_A - 2020-11-09 - Nodo K
 CLIE.   PROD.    CANT       P. U.
======= ===== ======== ===========
AAABBB      1       10        10.10
AAABBB      2        5        20.20
AAABBB      3        7        30.30
AAABBB      4       15        40.40
AAACCC      1      100        10.10
AAACCC      2       60        20.20
AAACCC      4        3        40.40
```

Ejemplo del archivo [**ped_B.txt**]

```
     2020-11-09 - Nodo C - Tipo_B
PRODUCTO  CANTI.  COD CLI  VALOR
======== ======== ======= =======
       1       10 AAAAAA    10.10
       5        5 AAAAAA    50.50
       3        7 AAABBB    30.30
       2       15 AAABBB    20.20
       4       13 AAADDD    40.40
       1       50 AAADDD    10.10
       2       60 AAADDD    20.20
       6      700 DDDAAA    60.60
       8      700 FFFAAA    80.80
       6      700 FFFBAA    60.60
       6       10 FFFBAA    60.60
```

En ambos archivos las 3 (tres) primeras líneas de texto que se ven están físicamente grabadas en ellos.

Glosario:

a- archivo de pedidos: [**ped_A.txt**], [**ped_B.txt**], etc.

Son generados por el sistema de ventas que corre en diversos servidores como medida pre emergente ante fallas del sistema. En estos archivos sus tres primeros registros (líneas de texto), siempre estarán presentes. El 3er registro determina el tamaño de cada campo.

b- pedidos binario: [**ped_A.bin**], [**ped_B.bin**], etc.

Responden a una estructura de información del tipo pedido ([**tPedido**]) cuyos miembros son (ver su declaración en el proyecto):

b.1- código de cliente: [**codClie**]

b.2- clave de producto: [**claProd**]

b.3- cantidad pedida: [**cantPed**]

b.4- precio unitario [**precUni**]

c- números de campos: array bidimensional de enteros, cada fila (vector de enteros) tiene un número indicando para cada campo del archivo de pedidos a qué miembro del tipo pedido (para generar el archivo pedido binario) corresponde.

Los distintos valores corresponden a: [**1**] **->** código de cliente, [**2**] **->** clave del producto, [**3**] **->** cantidad pedida, [**4**] **->** precio unitario

d- pedidos agrupados ([**ped_agrup.bin**])

Responde a la misma estructura de información que los de pedidos binario y resultan de la simple concatenación de los archivos de pedido binario

e- pedidos resumidos ([**ped_resum.bin**])

Responde a la misma estructura de información que los anteriores, con el agregado de un miembro más ([**tResumido**]), el número de registro ([**nroRegistro**])

Especificación funcional:

Cada archivo de pedidos debe ser convertido de texto a binario creando pedidos binario manteniendo el nombre (ver: [1]) para poder ser utilizados en otro módulo del sistema.

Así se obtendrán: [**ped_A.txt**] ==> [**ped_A.bin**], [**ped_B.txt**] ==> [**ped_B.bin**], etc.

A continuación (ver: [2]), se deberán unificar los archivos binarios en un único archivo de pedidos agrupados concatenándolos.

En un paso siguiente (ver: [3]), se procederá a generar un archivo de pedidos resumidos a partir del archivo de pedidos agrupados. Este archivo contendrá los pedidos de cada cliente ordenados por código de cliente / clave de producto, con el acumulado de la cantidad del pedido. Salvo en el caso que el precio unitario sea distinto (se puso en vigencia una nueva lista de precios a mitad del día / se pactó otro precio / etc.) con lo que se genera un nuevo registro para ese cliente / producto con el nuevo total acumulado de ese precio. Este archivo contendrá un número de registro que será un número secuencial comenzando en [1].

Especificación operativa:

Punto 1 a.- ([1])

El jefe de programción, sabiendo que hay más tipos de archivos ([**Tipo_A**] a [**Tipo_Z**]) agrega a la especificación que: en la 1er línea de cada archivo, hay que buscar la cadena de caracteres [**Tipo_**], y el carácter que está a continuación permite seleccionar del array bidimensional de enteros la fila que le corresponde ([**A**] = 1er fila, [**B**] = 2da fila, etc.). Esto se debe a que cuando se diseñó el sistema se previó tener identificadas zonas, distribuidores y vendedores, con los caracteres de la [**A**] a la [**Z**], [**a**] a la [**z**], etc., cosa que el analista pasó por alto (es un analista *junior* nuevo en la empresa), así que le define al programador que . . .

Se requiere una función que reciba

- nombre del archivo de pedidos (p. ej.: [**ped_A**] o [**ped_B**]) pero sin la extensión ([**.txt**])

- un array bidimensional de enteros con los números de campos

- un entero (filaTope)

Se debe leer el archivo de texto y generar el archivo pedidos binario.

Al leer el archivo, en la 1ra línea de texto, determina la '*letra*' a continuación del '*prefijo*' [**Tipo_**], y determina qué fila le corresponde en el array de enteros.

Con esa fila, se tendrá un vector de posiciones que vincula cada campo del archivo de texto con cada miembro de la variable del tipo registro.

- reglas para este punto: determinación de errores

#1 - si no se puede abrir el archivo de texto, devolverá [**-1**

#2 - si no se puede crear el archivo binario, devolverá [**-2**]

#3 – de producirse uno o ambos de los anteriores, devuelve la suma ([**-3**])

#4 - al leer el 1er registro de texto, se calcula si la fila que le corresponde excede el límite dado por [filaTope]. Si esto ocurriera devolverá [**-4**]

#5 – en caso de no ocurrir lo anterior, devolverá la cantidad de registros almacenados en el archivo binario

Se garantiza que los archivos de texto tienen al menos las 3 primeras líneas de texto.

Los tamaños de campo de los registros del archivo de texto se determinan con la 3ra línea del archivo de texto (donde comienza y termina cada '*subrayado*').

#6 – esta función debe cumplir *'silenciosamente*' su cometido, salvo que ocurran los errores antes indicados

La 2da línea de texto no es de utilidad y no se la deberá tener en cuenta.

La 3er línea de texto (en archivo de pedidos) determina el tamaño de cada campo en el mismo.

El jefe de programación hace énfasis en que en la fila de números de campos que le corresponde al archivo si los primeros cuatro enteros fueran [**2**], [**4**], [**3**] y [**1**],

- el 1er campo de archivo de pedidos se almacena en el 2do miembro de la variable del tipo pedido (clave de producto)

- el 2do campo en el 4to miembro (precio unitario)

- el 3er campo en el 3er miembro (cantidad pedida)

- el 4to campo en el 1er miembro (código de cliente)

. . . tras lo cual se podrá grabar ese registro en pedidos binario.

Punto 1 b.- ([**2**])

Se requiere una función que permita concatenar los archivos de pedidos binario en el archivo pedidos agrupados. La función recibirá el nombre (completo) del archivo de salida, y hasta cuatro nombres de archivos de entrada (el jefe de programación planea modificar luego esta función para que admita una cantidad variable de archivos). La función devolverá [**-1**] si no puede abrir el archivo de salida, [**-2**] si no puede abrir el 1er archivo de entrada, [**-3**] si no puede abrir el 2do archivo de entrada, etcétera. Antes de terminar, si se produce un error con alguno de los archivos de entrada, eliminará el archivo de salida y devolverá el indicador correspondiente. En caso de poder cumplir su

cometido, devolverá la cantidad de registros grabados. Respetar lo indicado en la especificación funcional.

Punto 1 c.- ([3])

Se requiere una función que genere el archivo pedidos resumidos a partir de pedidos agrupados. La función recibirá en su 1er argumento el nombre del archivo de salida y en el 2do argumento el de entrada. Para esto se dispone de la biblioteca que permite el uso del tipo de dato pila y sus primitivas. Una vez ordenados y acumulados los registros al generar el archivo de salida se asignará el miembro número de registro con un número secuencial. En caso de no poder abrir el archivo de salida, devolverá [**-1**], si no puede abrir el archivo de entrada, devolverá [**-2**], si no pudiera abrir ninguno, devolverá la suma de los indicadores. Si pudo abrir el archivo de salida, pero no el de entrada deberá eliminar el de salida (quedó vacío). En el caso de cumplir con su cometido, devolverá la cantidad de registros grabados. Tener en cuenta la especificación funcional.

NO DEBE USAR ARRAYS AUXILIARES PARA COPIAR LAS CADENAS.

Solamente si debe crear el path de un archivo (Punto 1 a.-)

PUEDE USAR FUNCIONES DE BIBLIOTECA.

(<stdlib.h>, <ctype.h>, <string.h>, <stdio.h>, . . . las estándar)

NO DEBE USAR SUBÍNDICES NI PUNTERO CON DESPLAZAMIENTO VARIABLE.

NO PUEDE USAR LAS FUNCIONES DE BIBLIOTECA:

`fscanf` ni `sscanf` ni `sprintf`

Punto 2

Desarrollar una clase **`Punto`**, con coordenadas **`x`** e **`y`** de tipo **`double`**.

Desarrollar una clase **`Recta`**, conformada por dos puntos de la clase anterior.

Debe resolver las clases de manera que compile y ejecute correctamente las funciones provista en **`main`** (ver "condición mínima para no reprobar" en la página 2)

Ecuación de la distancia de un punto a una recta:

Dada una recta que pasa por 2 puntos **$P_1 = (x_1, y_1)$** y **P2 = (x2, y2)**, la distancia del punto **(x0, y0)** a la recta está dada por:

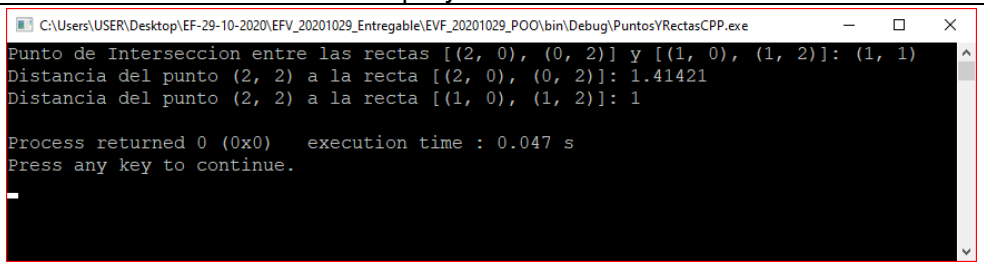
$$\frac{|(y_2 - y_1)x_0 - (x_2 - x_1)y_0 + x_2y_1 - y_2x_1|}{\sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}}$$

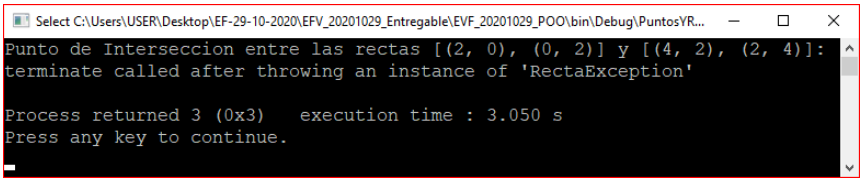
Ecuación de la intersección de 2 rectas:

Dada la recta: $R1 = [(x_1, y_1), (x_2, y_2)]$
y la recta: $R2 = [(x_3, y_3), (x_4, y_4)]$,
su intersección, el punto: $P = (x, y)$ está dado por:

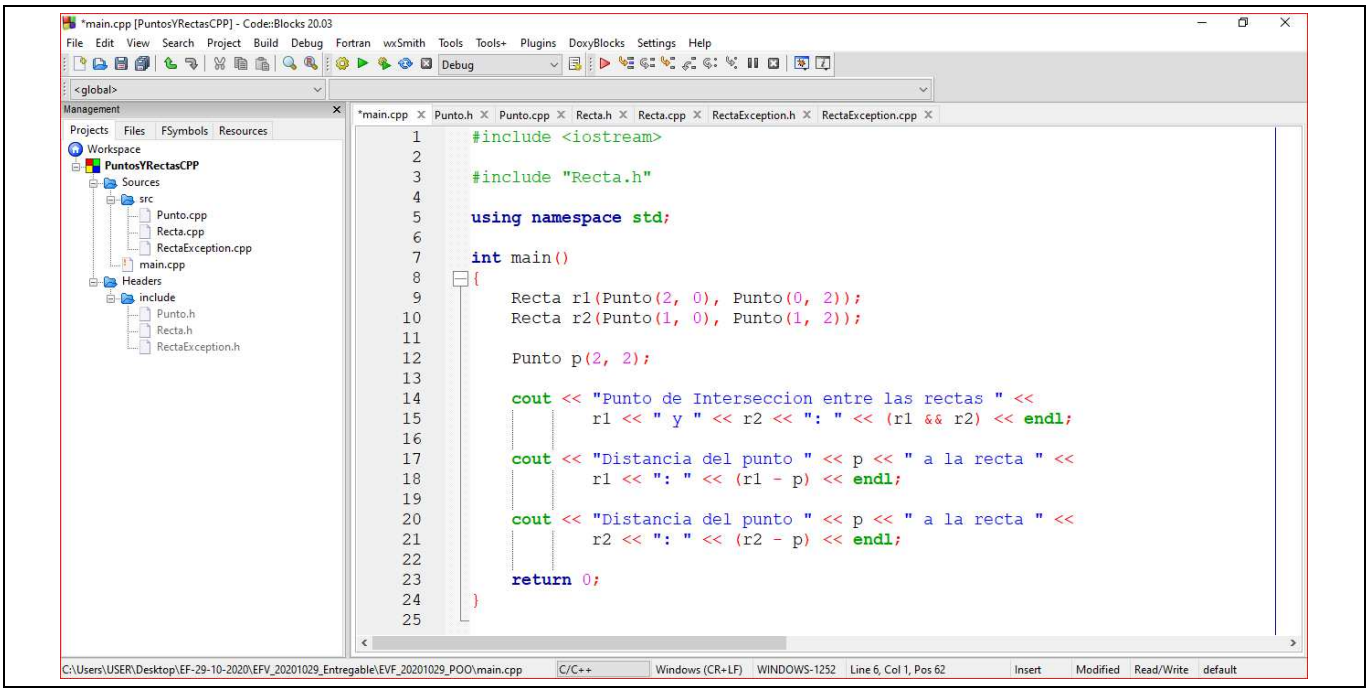
$$(x, y) = \frac{(x_1y_2 - y_1x_2)(x_3 - x_4) - (x_1 - x_2)(x_3y_4 - y_3x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)} \quad , \quad \frac{(x_1y_2 - y_1x_2)(y_3 - y_4) - (y_1 - y_2)(x_3y_4 - y_3x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)}$$

Salida del proyecto de POO – C++





UNLaM
Dto. Ingeniería e Investigaciones Tecnológicas



Plazo de entrega: el miércoles 11/11/2020 – 08:00 horas.