Metodología de la Programación

PRÁCTICA FINAL

Grupo C

Universidad de Granada
Departamento de Ciencias de la Computación
e Inteligencia Artificial

Metodología de la Programación (2015/16)
1º Grado en I.I. (Evaluación única final)
Junio de 2016

DECSAI

Normas para la realización del examen: **Duración: 3.5 horas**

- **El único material permitido durante la realización del examen es un bolígrafo azul o negro.**
- **Debe disponer de un documento oficial que acredite su identidad a disposición del profesor.**
- **No olvide escribir su nombre completo y grupo en todos y cada uno de los folios que entregue.**

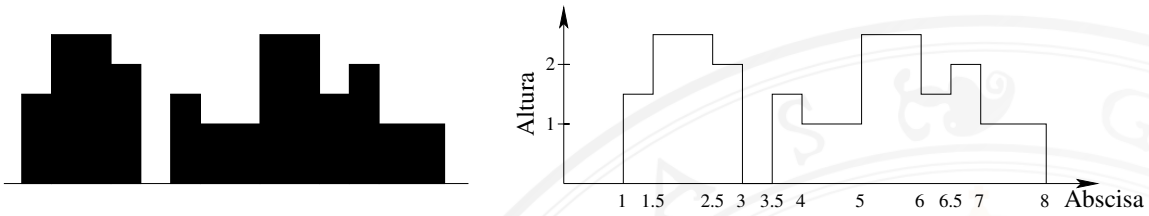
◁ Ejercicio 1 ▷ Vectores-C [0.75 puntos]

Implemente una función `MezclarUnico` de tipo `int` que recibe como entrada dos vectores-C de datos de tipo `double` y los mezcla en un tercer vector. Tenga en cuenta que:

- Los vectores de entrada están ordenados y sin valores repetidos.
- El vector de salida tendrá los elementos ordenados y sin repetidos.
- Puede asumir que el vector de salida tiene capacidad suficiente para todos los elementos.
- La función devuelve un entero que es el número de elementos que contiene el vector de salida. Nota: Será menor o igual que la suma de los de entrada.

◁ Ejercicio 2 ▷ SkyLine: clase [2.25 puntos]

Se define un SkyLine como la silueta urbana que refleja la vista general de los edificios de una ciudad. Se propone crear un programa para procesar SkyLines simplificados, donde los edificios no son más que un rectángulo que se eleva desde una línea base horizontal. La siguiente figura muestra un ejemplo:



donde podemos ver que se puede codificar guardando los valores de abscisas y alturas de cada cambio de altura.

Se desea crear un programa para procesar SkyLines. Para ello, se propone la siguiente representación:

```
class SkyLine {
    double *abscisas;  // abscisas[i] > abscisas[i-1]
    double *alturas;   // alturas[i]>=0 && alturas[n-1] == 0
    int n;             // tamaño de los vectores anteriores.
  public:
     // ... interfaz pública de la clase
};
```

En el ejemplo de la figura anterior, el valor de `n` sería 11 y los dos vectores de este tamaño contendrían:

- Abscisas: *1, 1.5, 2.5, 3, 3.5, 4, 5, 6, 6.5, 7, 8*.
- Alturas: *1.5, 2.5, 2, 0, 1.5, 1, 2.5, 1.5, 2, 1, 0*.

Defina las siguientas funciones:

1. *(0.5 puntos)* El contructor por defecto y el destructor. El constructor por defecto crea el SkyLine nulo, representado con todos sus campos a cero.
2. *(0.75 puntos)* Un constructor para un edificio. Recibe la descripción de un edificio compuesta por tres valores: dos abscisas y una altura; construye el skyline correspondiente.
3. *(1 punto)* El constructor de copia y operador de asignación.

◁ Ejercicio 3 ▷ Skyline: miembro público [0.5 puntos]

Considere la clase `Skyline` anterior. Se decide añadir una función para calcular la altura del skyline en cualquier punto. Implemente una función miembro *Altura* que recibe una abscisa y devuelve la altura en ese punto. Tenga en cuenta que la altura fuera del skyline es cero.

◁ Ejercicio 4 ▷ Skyline: Sobrecarga de operadores [1.5 puntos]

Considere la clase `Skyline` anterior. Implemente las siguientes funciones miembro:

1. *(0.75 puntos)* Sobrecarga del operador de suma para poder obtener el skyline que corresponde a superponer dos skylines. El algoritmo consiste en crear las abscisas como la mezcla de los dos vectores abscisas de entrada y situar las alturas como el máximo de entre los dos skylines de entrada. Puede considerar resueltas y usar las funciones de los ejercicios 1 y 3.
2. *(0.25 puntos)* Sobrecarga del operador de salida << para poder obtener en un flujo de salida el skyline en formato texto. En concreto, deberá aparecer el número de abscisas y, en distintas líneas, cada una de las parejas que componen los valores de abscisa y altura.
3. *(0.5 puntos)* Sobrecarga del operador de entrada >> para poder obtener desde un flujo de entrada el skyline en formato texto. Suponga que el formato está compuesto por el número de abscisas y cada una de las parejas que componen los valores de abscisa y altura; todos los valores separados por *"espacios blancos"*.

◁ Ejercicio 5 ▷ Skyline: Funciones externas y ficheros [1 punto]

Se desea añadir la posibilidad de cargar/salvar el contenido de un skyline desde/a un fichero. Para ello, se establece un formato de texto que consiste en:

- Una cadena mágica `"MP-SKYLINE-1.0"` y un salto de línea para poder distinguir que es un fichero tipo skyline.
- El skyline en formato de texto tal como se indica en el ejercicio 4.

Considere que tiene disponibles las funciones del ejercicio 4. Implemente las siguientes funciones externas:

1. Una función `Cargar` que recibe un nombre de archivo y un objeto de tipo `SkyLine` y carga en éste el contenido del archivo. Devuelve si ha tenido éxito.
2. Una función `Salvar` que recibe un nombre de archivo y un objeto de tipo `Skyline` y lo salva en el archivo. Devuelve si ha tenido éxito.

◁ Ejercicio 6 ▷ Skyline: uso de la clase [1 punto]

Considere la clase `Skyline` anterior. Escriba un programa —*montar.cpp*— que lee desde la entrada estándar una secuencia de edificios y escribe en un fichero el skyline resultado. Tenga en cuenta que:

- En la entrada, cada edificio se especifica como dos valores que indican las abscisas de la base más un tercero para indicar la altura. La entrada termina cuando se introduce una base con dos valores idénticos.
- El nombre del fichero resultado se especifica en la línea de órdenes. Puede suponer resuelta y usar la función de salvado del ejercicio 5.

◁ Ejercicio 7 ▷ Skyline: Modificando la parte interna [1 punto]

El algoritmo de suma de dos skylines propuesto da lugar a un resultado que puede contener varios valores de altura consecutivos idénticos. En este caso, se puede simplificar manteniendo únicamente el primero de ellos. Escriba una función miembro *Simplificar* privada para minimizar el número de puntos almacenados en el skyline y así el espacio que ocupa. Implemente esta función miembro tal y como aparecería en el archivo *cpp*.

◁ Ejercicio 8 ▷ Skyline: módulos [1 punto]

Considere la clase `Skyline` anterior. Escriba el contenido del archivo de cabecera *skyline.h*, donde aparece todo lo necesario para usar la clase, sin incluir ninguna definición de función (suponga que estarán en *skyline.cpp*).

◁ Ejercicio 9 ▷ Skyline: makefile [1 punto]

Considere los archivos que implementan el tipo `skyline` así como el programa *montar.cpp* anterior. Escriba el archivo *Makefile* que permite compilar el programa. Incluya las correspondientes dependencias, así como un objetivo *clean* que permite limpiar los archivos intermedios.

Normas para la realización del examen: **Duración: 1 hora**

- **El único material permitido durante la realización del examen es un bolígrafo azul o negro.**
- **Debe disponer de un documento oficial que acredite su identidad a disposición del profesor.**
- **No olvide escribir su nombre completo y grupo en todos y cada uno de los folios que entregue.**

Se pretende implementar una clase que permita representar de forma eficiente matrices dispersas, donde sólo un número relativamente bajo de valores son distintos de cero. En este caso sólo es necesario almacenar información de aquellos valores significativos, no cero. Para cada valor significativo hay que almacenar: **fila**, **columna** (de tipo *int*) y **valor** (de tipo *double*). Esta información puede gestionarse de la forma que consideréis oportuna (mediante estructura o clase; asumamos que su nombre es **Valor** y están disponibles ya todos los métodos de asignación y acceso a sus datos). De esta forma se propone la siguiente representación para la clase:

```
class MatrizDispersa {
  private:
    int nfilas;           // Número de filas
    int ncolumnas;        // Número de columnas
    Valor *valores;       // Vector-C no ordenado, solo valores significativos
    int numeroValores;    // Número de valores significativos almacenados
  public:
     // ... interfaz pública de la clase
 };
```

NOTA: los objetos de tipo **Valor** no tienen por qué estar almacenados por orden de filas y columnas en el vector-C **valores**. A modo de ejemplo consideremos la matriz dispersa a la derecha. Su tratamiento requiere de un vector-C con 4 datos de tipo **Valor**, con la información:

fila $= 1$, columna $= 1$, valor $= 1.0$
fila $= 2$, columna $= 2$, valor $= 2.0$
fila $= 3$, columna $= 3$, valor $= 3.0$
fila $= 4$, columna $= 4$, valor $= 4.0$

1.0	0.0	0.0	0.0
0.0	2.0	0.0	0.0
0.0	0.0	3.0	0.0
0.0	0.0	0.0	4.0

◁ Ejercicio 1 ▷ Vectores-C y función externa a la clase

Implemente la función externa **combinarSuma** que reciba como entrada dos vectores-C de datos de tipo **Valor** y produce como resultado un tercer vector del mismo tipo (puede asumir que el vector de salida tiene espacio suficiente para almacenar todos los valores que se produzcan). Tenga en cuenta que:

- el valor de una posición se determina de la siguiente forma: si un valor aparece sólo en un vector de entrada, entonces mantiene su valor; si aparece en ambos vectores, entonces el valor almacenado será la suma de ambos (incluso si la suma es 0).
- la función puede contener los argumentos que considere necesarios para su funcionamiento.
- la función devolverá el número de datos de tipo **Valor** almacenados en el vector-C de resultado.
- puede implementar funciones auxiliares si se estima oportuno.

◁ Ejercicio 2 ▷ Métodos básicos de la clase

Defina los siguientes métodos para la clase **MatrizDispersa**:

1. **constructor por defecto** y **destructor**. El constructor por defecto crea una matriz sin elementos (0 filas, 0 columnas, 0 valores).
2. **constructor con parámetros**. Recibe dos enteros con las dimensiones de la matriz, junto con tres vectores-C y un entero que indica sus tamaños (total: 6 parámetros). Cada posición de los vectores corresponde a una tripleta fila/columna/valor de la matriz. Se asume que no habrá dos tripletas que correspondan al mismo par fila/columna, que las parejas fila/columna están dentro de las dimensiones indicadas y que ningún valor es igual a cero.
3. *(1 punto)* **constructor de copia** y **operador de asignación**.

◁ Ejercicio 3 ▷ Obtención y asignación de valor

Se decide añadir a la clase **MatrizDispersa** los métodos:

- **obtenerValor**: permite recuperar el valor asociado a una determinada posición. Recibe la fila y columna de interés; devuelve 0 en caso de no haber valor significativo en esa posición ($(1, 2)$ en el ejemplo anterior) o el valor almacenado (4.0 para posición $(4, 4)$). Se supone que los valores de fila y columna siempre serán válidos.

- **asignarValor**: permite modificar el contenido de la matriz. Recibe la fila y columna de interés y el valor de dicha posición. Si existe un valor para dicha posición, lo sustituye (a menos que el nuevo valor a asignar sea 0, en cuyo caso se debe reducir el número de valores significativos de la matriz). Si se trata de una posición no asignada previamente, se agrega el nuevo valor (siempre que sea distinto de 0).

◁ Ejercicio 4 ▷ Sobrecarga de operadores

Implemente la siguiente funcionalidad para la clase **MatrizDispersa**:

1. **operador de suma** para poder obtener la suma de dos matrices dispersas. Puede usar las funciones pedidas en los ejercicios previos (**combinarSuma** y **asignarValor**). Se asume que las matrices sumadas tienen el mismo número de filas y columnas.
2. **operador de salida** << para poder obtener en un flujo de salida el contenido de la matriz en formato texto. En concreto, mostrará: número de filas, número de columnas, número de valores significativos y, en distintas líneas, el contenido de cada objeto de tipo **Valor**: fila, columna y valor.
3. **operador de entrada** >> para poder obtener desde un flujo de entrada el contenido de un objeto. Se asume el mismo formato indicado anteriormente.

◁ Ejercicio 5 ▷ Funciones externas y ficheros

Se desea añadir la posibilidad de cargar y salvar el contenido de una matriz dispersa desde fichero. Para ello, se establece un formato de texto que consiste en:

- una cadena mágica `"MP-MATRIZDISPERSA-1.0"` y un salto de línea (para poder distinguir que es un fichero usado para almacenar información de objetos de la clase de interés).
- el contenido de la matriz en formato de texto, tal como se indica en el ejercicio 4.

Considerando que tiene disponibles los métodos de los ejercicios previos implemente las siguientes funciones externas:

1. función **cargar**, que recibe un nombre de archivo y un objeto de tipo **MatrizDispersa** y carga en éste el contenido del archivo. Devuelve si la operación ha tenido éxito.
2. función **salvar**, que recibe un nombre de archivo y un objeto de tipo **MatrizDispersa** y lo salva en el archivo. Devuelve si la operación ha tenido éxito.

◁ Ejercicio 6 ▷ Ampliación de la clase [1 punto]

Implemente un nuevo constructor de la clase **MatrizDispersa** que reciba como parámetros las filas y columnas y un tercer parámetro opcional, que es un puntero a los datos. Este puntero tiene 0 como valor por defecto, en cuyo caso representa una matriz con todos los valores a 0.

◁ Ejercicio 7 ▷ Uso de la clase [1 punto]

Escriba un programa (**principal.cpp**) que lea desde la entrada estándar una secuencia con el contenido de la matriz y genere el archivo correspondiente. El nombre del archivo se especifica en la línea de órdenes. Recuerde que puede asumir resueltas y disponibles las funciones y métodos mencionados en los ejercicios previos.

◁ Ejercicio 8 ▷ Método de poda [1 punto]

Implemente un método llamado **podar**, que produce un nuevo objeto de la clase, mediante la siguiente simplificación: todos aquellos valores que estén por debajo de un cierto valor umbral se eliminan (reduciéndose así el número de valores almacenados en la matriz). Se valorará la eficiencia del método.

◁ Ejercicio 9 ▷ Organización en módulos y Makefile [1 punto]

1. *(0.5 puntos)* Escriba el contenido del archivo de cabecera **matrizDispersa.h**, donde debe aparecer todo lo necesario para usar la clase, sin incluir ninguna definición de función (suponga que estarán en **matrizDispersa.cpp**).
2. *(0.5 puntos)* Considere los archivos que implementan el tipo **MatrizDispersa**, así como el programa **principal.cpp** indicado con anterioridad. Escriba el archivo **Makefile** que permita la compilación del programa. Incluya las dependencias necesarias y un objetivo **clean** que permita eliminar los archivos intermedios generados en el proceso de compilación.