Nombre: _____
Código: _____

1. Introducción

Este parcial consta de 4 puntos cada uno con el mismo valor porcentual para el cálculo de la nota, pero cada punto tiene su propia descriminación de porcentajes según el trabajo a realizar.

1.1. Enlaces

Las salas permanecerán bloqueadas a internet, excepto a los siguientes recursos:

Lugar	**Enlace**
Riouxsvn	`https://riouxsvn.com`
C++ Reference	`http://en.cppreference.com/w/`

1.2. Directorio `parcial02`

Cuando el parcial se complete se debe obtener una lista de ficheros semejante a la siguiente:

Figura 1: Jeraquía de directorios `parcial02`

2. Preguntas

1. (25 points) **Creación de eco**
   (directorio: `EcoA`) En el fichero `eco.h` se encuentra la descripción de las funciones que se van a implementar y utilizar:

```
int*
creacionDeEco(const int arreglo[], const int longitud,
        const int restraso, float factor);

void
imprimirEco(const int eco[], const int longitud);
```

   a) (8 puntos) **Descripción e implementación de la función `creacionDeEco`**
   (fichero: `eco.cpp`) Esta función recibe un `arreglo` de valores `int` de un `longitud`, un `retraso` de donde se comienza a generar el eco y el `factor` de disminución del eco ($factor \leq 1{,}0$).
   Suponga que el arreglo tiene los siguientes valores:

```
int arreglo[5] =  100, 200, 1000, -150, -350 ;
```

   Con un restraso de 3 y un factor de 0,6 se aplica así:

```
100, 200, 1000 -150, -350
                 100,  200
               * 0.6
                 60,   120
+
------------------------
100, 200, 1000, -90, -230
```

   El arreglo de salida es igual a:

```
100,200,1000,-90,-230
```

   b) (7 puntos) **Descripción e implementación de la función `imprimirEco`**
   (fichero: `eco.cpp`) Esta función recibe un arreglo de enteros que representa un eco y imprime los valores separados por una coma y un espacio, excepto el último elemento:

c) (7 puntos) **Descripción e implementación del programa principal `procesarecos.cpp`**
(fichero `procesarecos.cpp`) genera un ejecutable `procesarecos.exe` que recibe un entrada de la siguiente forma:

```
3
5 3 0.6
100 200 1000 -150 -350
6 2 0.7
12 213 2343 -123 -134 10
5 1 0.8
134 1234 13 234 -978
```

Donde el primer valor es el número de elementos que se van a procesar: 3 generaciones de ecos. La segunda línea tiene tres valores: el número de valores que contiene el eco 5, el restraso que se va aplicar 3 y el último el factor 0,6. El resto de las líneas sigue en el mismo formato hasta completar los ecos a procesar.

La anterior entrada genera la siguiente salida:

```
100, 200, 1000, -150, -350
100, 200, 1000, -90, -230
12, 213, 2343, -123, -134, 10
12, 213, 2351, 26, 1506, -76
134, 1234, 13, 234, -978
134, 1341, 1000, 244, -790
```

d) (3 puntos) **Implementación del `Makefile`**
Implementar el fichero `Makefile` que compile el proyecto anterior.

2. (25 points) **Clases, objetos en C++**

   (directorio: `DeviceA`) En la figura 2 se observa una diagrama de clases en UML, en el se observa la clase `Device` que representa el valor almacenado por un dispositivo. Su subclases `DevByFive` y `DevByTen` múltiplican el valor del `factorOne` en 5 y 10 respectivamente. La subclase `Device2` tiene otro atributo llamado `factorTwo`.

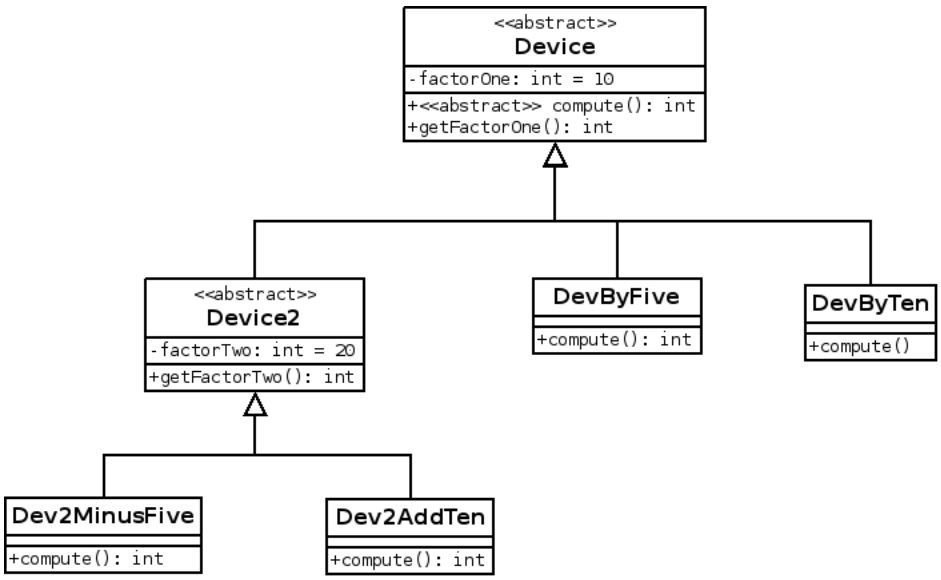

Figura 2: Clases, objetos en C++

Para la subclase de `Device2`: `Dev2MinusFive` el computo se hace sumando a `factorOne` el valor obtenido de restarle 5 al `factorTwo`. Para la sublclase de `Device2`: `Dev2AddTen` el computo se hace sumando a `factorOne` el valor obtenido de sumar 10 a `factorTwo`.

a) (11 puntos) En este punto de debe implementar la anterior de jeraquía de clases en los ficheros: `device.h` y `device.cpp`.

b) (11 puntos) Luego se procede a utilizarlos en un programa que se va a llamar: `deviceuse` (fichero: `deviceuse.cpp`) y que genera un ejecutable `deviceuse.exe`.

   Este programa (`deviceuse.exe`) lee de la entrada estandar un entero que representa el número de dispositivos que se van a leer. Luego lee el número de dispositivos tipo `Device` (`DevByFive` y `DevByTen`). Luego lee los dispositivos tipo `Device` y luego lee los restantes tipo `Device2` (`Dev2MinusFive` y `Dev2AddTen`).

   Un posible entrada es la siguiente:

```
5
3
5 10
10 20
5 10
5 2 10
10 2 5
```

En ella se observa que la primera línea se tiene que se van a leer 5 dispositivos. En la segunda línea se encuentra un 3 que indica el númro de dispositivos tipo `Device`. Las tres siguiente líneas: lee un dispositivo `DevByFive` con valor inicial 10, lee un dispositivo `DevByTen` con valor incial 20, lee un dispositivo `DevByFive` con valor inicial 10. Las dos últimas se encarga de leer dos dispositivos de tipo `Device2`: el primero un dispositivo de tipo `Dev2MinusFive` con valores iniciales 2 y 10, lee un dispositivo `Dev2AddTen` con un valore iniciales 2 y 5.

El resultado con los valores anteriores produce la siguiente salida:

```
324
```

Todos los valores deben ser almacenados en el *heap* a través de un apuntador definido de la siguiente manera:

```
1 Device *arreglo;
```

c) (3 puntos) Crear un fichero `Makefile` que se encarga de compilar el proyecto.

3. (25 points) **Manejo de intensidad**
(directorio: `IntensidadA`) Escriba una procedimiento llamado revisarIlluminacia:

```
void
revisarIluminancia(float pRojo, float pVerde,
                   int rojo, int verde, int azul);
```

Los dos primeros valores son valores flotantes que son menores a uno, y ambos deben tienen la siguiente restricción:

$$pRojo + pVerde \leq 1$$

Los valores representa el porcentaje encontrado de rojo y de verde. El porcentaje de Azul se obtiene de la siguiente fórmula:

$$pAzul = 1 - (pRojo + pVerde)$$

La función calcula el promedio ponderado de los colores que es la suma de todos los porcentaje de participación de los colores. El porcentaje de participación se calcula así:

$$pColor * color$$

El programa imprime lo siguiente si el valor resultante es:

- Menor de 10. “Oscuro.”
- Entre 50 y 200. “Bien.”
- Mayor 250. “Transparente.”

En cualquier otro caso no se imprime nada.

a) (11 puntos) `Descripcion e emplementación de la función revisarIluminancia`
Se escriben dos ficheros correspondientes a la interfaz y a la implementación: `iluminanciaA.h` y `iluminanciaA.cpp`.

b) (11 puntos) `Programa principal`

Se escribe un fichero que lee los siguientes valores de entrada y utilizando la función `revisarIlumnancia`:

```
0.1 0.8 100 89 90
0.2 0.5 100 100 400
0.9 0.01 1 5 6
```

Produzca la siguiente salida:

```
Bien.
Transparente.
Oscuro.
```

c) (3 puntos) `Makefile`

Escriba el fichero `Makefile` que permita crear el anterior proyecto.

4. (25 points) **Baldosas *strikes back again***

   (directorio: `BaldosasA`) Una fabrica de baldosas tiene muchos patrones de baldosas que puede producir en serie: los patrones se representa con letras mayúsculas *M* y letras minúsculas *m*. Los patrones posibles son: *MM*, *Mm*, *mM* y *mm*. La fabrica tiene una super máquina que produce un lote de baldosas de un patrón particular llamada Troqueladoramatic, cariñosamente apodada troquic. Troquic no se le ha hecho mantenimiento hace muchos años y ya no produce un solo patrón como se espera sino que accidentalmente puede produccir patrones diferentes. Se ha diseñado una máquina que es capaz de ver las baldosas producidas por Troquic y generar una cadena de caracteres par con los patrones de las baldosas condificados en ASCII.

   En la producción de hoy: troquic ha sido configurada para producir baldosas *mM*.

   Escribir un programa que se llame (`baldosas.exe`), su fichero fuente se llama `baldosas.cpp`. Este programa recibe un entrada en el siguiente formato:

```
aBnJoQlMzHoPaVvNDFaB
aBbCcDdEfG
zBxXiKlKlKlMoMoMoM
aBiBiBcC
AaAaaAaA
aAbBcDdD
IoJkKkKk
KKKKKKKK
```

   produce la siguiente salida:

```
aBnJoQlMzHoPaVvNDFaB total: 10 buenas: 9 malas: 1
aBbCcDdEfG total: 5 buenas: 5 malas: 0
zBxXiKlKlKlMoMoMoM total: 9 buenas: 9 malas: 0
aBiBiBcC total: 4 buenas: 4 malas: 0
AaAaaAaA total: 4 buenas: 2 malas: 2
aAbBcDdD total: 4 buenas: 4 malas: 0
IoJkKkKk total: 4 buenas: 0 malas: 4
KKKKKKKK total: 4 buenas: 0 malas: 4
```