Programación 2

Examen de teoría (junio 2019)

29 de mayo de 2019

Instrucciones

- **Duración: 3 horas**
- El fichero del primer problema debe llamarse `got.cc`. Para el segundo problema es necesario entregar cuatro ficheros, llamados `Station.cc`, `Station.h`, `Region.cc` y `Region.h`. Pon tu DNI y tu nombre en un comentario al principio de todos los ficheros fuente.
- La entrega se realizará como en las prácticas, a través del servidor del DLSI (`http://pracdlsi.dlsi.ua.es`), en el enlace `Programación 2`. Puedes realizar varias entregas, aunque sólo se corregirá la última.
- En la página web de la asignatura `http://www.dlsi.ua.es/asignaturas/p2` tienes disponibles algunos ficheros que te pueden servir de ayuda para resolver los problemas del examen, así como el apartado `Reference` de la web `www.cplusplus.com`.

Problemas

1. **(6 puntos)**

   El invierno ya ha llegado y te has alistado para defender Poniente frente al ejército del Rey de la Noche.[1] Por suerte para ti, llegaste armado sólo con tus conocimientos de informática y te han declarado no apto para el combate, así que deberás ayudar a Samwell Tarly a organizar las tropas para la batalla de esta noche.

   Las tropas disponibles deberán dividirse en pelotones de 50 soldados. En cada pelotón todos los soldados deben ser del mismo tipo (caballería, infantería, arqueros, artillería,...) y todos deben pertenecer a la misma familia (Stark, Cassel, Dothraki,...) para evitar conflictos. Al dividir en grupos de 50, si queda algún pelotón con un número inferior a éste, se creará un pelotón nuevo sólo si hay al menos 30 soldados. En caso contrario, se unirán al llamado pelotón mixto (`Mixed`). Este pelotón puede tener más de 50 unidades, así como miembros pertenecientes a distintas familias y tipos.

   Los datos sobre las tropas con las que contamos están almacenados en un fichero binario, llamado `troops.dat`, que contendrá un conjunto de registros con la siguiente estructura:

   ```
   char family[40];    // Nombre de la familia a la que pertenecen los soldados
   char type[20];      // Tipo de soldados: cavalry (caballería), infantry (infantería), ...
   unsigned int units; // Número de soldados (unidades) disponibles
   ```

   Por otra parte, los últimos enfrentamientos han provocado numerosas bajas entre nuestras filas que no ha dado tiempo a registrar en ese fichero. Han llegado varios cuervos con los informes de bajas y hemos tenido que codificarlos en un fichero de texto, llamado `losses.txt`, con una línea para cada batalla siguiendo este formato:

   ```
   Location|family-type:units|family-type:units|...
   ```

   donde `Location` es el lugar de la batalla, mientras que `family`, `type` y `units` tienen el mismo significado que el descrito anteriormente para el fichero binario.

   Tu labor ahora es escribir un programa que lea el fichero binario para calcular el número de tropas disponibles de cada familia y tipo, actualizar este número restando todas las bajas registradas en el fichero de texto,[2] agrupar los soldados en pelotones y, finalmente, generar un informe por pantalla como el que se muestra en el siguiente ejemplo:[3]

[1] Este enunciado está 99% libre de *spoilers*.

[2] Si no se puede restar, porque en el fichero binario no existe esa familia, o no existen soldados de ese tipo para esa familia, o el número de bajas es mayor que el de soldados, se deberá mostrar el error `Wrong troop` y seguir procesando la línea.

[3] Si no se puede abrir alguno de los dos ficheros, se debe mostrar un error y finalizar el programa. Se asume que el formato de los dos ficheros será siempre correcto, por lo que no es necesario comprobarlo.

- Contenido de `troops.dat`:[4]

```
{ "Mormont", "infantry", 120 }
{ "Mormont", "artillery", 50 }
{ "Stark", "infantry", 250 }
{ "Mormont", "infantry", 80 }
{ "Stark", "artillery", 30 }
{ "Cassel", "infantry", 70 }
{ "Dothraki", "cavalry", 125 }
```

- Contenido de `losses.txt`:

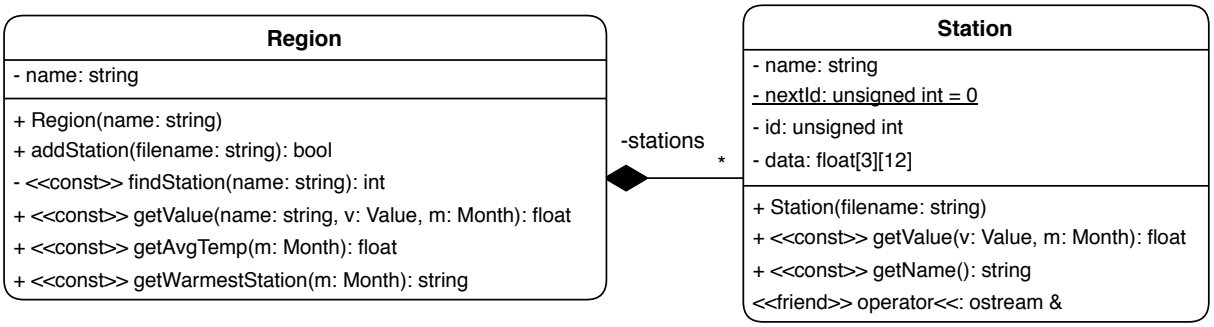
```
Iron Isles|Dothraki-cavalry:72|Stark-infantry:30|Stark-cavalry:120
Dorne|Mormont-infantry:60|Mormont-artillery:15|Stark-infantry:115|Cassel-infantry:60
```

- Salida del programa por pantalla, mostrando el informe con el reparto en pelotones:

```
Wrong troop
1. Mormont|infantry|50
2. Mormont|infantry|50
3. Mormont|infantry|40
4. Mormont|artillery|35
5. Stark|infantry|50
6. Stark|infantry|50
7. Stark|artillery|30
8. Dothraki|cavalry|50
9. Mixed|18
```

El orden en el que se muestran los pelotones configurados es indiferente. Lo único importante es que el pelotón `Mixed` siempre deberá aparecer el último (aunque si no tiene soldados, no deberá mostrarse) y no se deberá mostrar su tipo (`type`), ya que sus soldados pueden pertenecer a diferentes tipos.

2. **(4 puntos)**



La Agencia Española de Meteorología (AEMET) nos ha encargado implementar un programa para obtener estadísticas sobre la temperatura en España. Por cada estación meteorológica se nos proporciona un fichero de texto que contiene la media mensual de diferentes datos climatológicos calculada entre 1981 y 2010. Ejemplo:

```
Alicante
January 11.7 17.0 6.3
February 12.3 17.6 7.1
March 14.2 19.6 8.9
April 16.1 21.3 10.9
May 19.1 24.1 14.1
June 22.9 27.8 18.1
```

[4] Se trata de un fichero binario, por lo que éste no es el formato real del fichero, sino que cada línea representa la información almacenada en uno de los registros: `family`, `type` y `units`.

```
July 25.5 30.3 20.7
August 26.0 30.8 21.2
September 23.5 28.5 18.5
October 19.7 24.9 14.5
November 15.4 20.5 10.3
December 12.6 17.7 7.4
```

La primera línea del fichero contiene el nombre de la estación, mientras que el resto de líneas contienen cada uno de los doce meses del año, con su temperatura media (`T`), temperatura máxima (`TM`) y temperatura mínima (`Tm`) asociada.

Como puede verse en el diagrama, debemos implementar dos clases: `Station` (estación) y `Region` (región). Cada estación contiene un nombre (`name`), un identificador (`id`) y una matriz que almacenará los datos climatológicos (`data`). El constructor de `Station` recibe el nombre de un fichero que deberemos leer,[5] guardando el nombre de la estación (primera línea) y los datos de temperatura, temperatura máxima y temperatura mínima. Por tanto, `data` es una matriz que contiene 3 filas (una por cada uno de estos tres datos) y 12 columnas (una para cada uno de los meses del año). Si el constructor no puede abrir el fichero, debe lanzar una excepción (sin llegar a mostrar ningún mensaje de error). En el constructor, el `id` asignado a la estación se corresponderá con el valor de `nextId`, que deberá ser incrementado para la siguiente llamada al constructor.

Para facilitar el acceso a la información de la matriz, debemos declarar los siguientes tipos enumerados en el fichero `Station.h`:

```
enum Value {T,TM,Tm};
enum Month {Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec};
```

Con estos nuevos tipos se puede acceder, por ejemplo, a la temperatura máxima del mes de abril como `data[TM][Apr]`.[6] El método `getValue` de la clase `Station` recibe los parámetros de tipo `Value` y `Month`, devolviendo el valor correspondiente de la matriz `data`. El operador salida de `Station` sólo debe imprimir el nombre de la estación seguido de su identificador entre paréntesis. Por ejemplo: `Alicante (0)`.

La clase `Region` tiene un constructor que recibe un nombre (`name`). Por ejemplo: "Comunitat Valenciana". El método `addStation` debe crear y añadir una nueva estación a partir del nombre de fichero que recibe. Si ya existía una estación con ese nombre, se actualizarán sus datos meteorológicos pero no se añadirá una nueva. Si se produce una excepción cuando se construye la estación, este método debe capturarla y mostrar el mensaje de error `Error opening file`, devolviendo `false`. En caso contrario devolverá `true`. El método `findStation` debe buscar el nombre de la estación en el vector y devolver su posición o `-1` si no la ha encontrado. El método `getValue` recibirá el nombre de una estación, un valor y un mes. Este método debe usar `findStation` para buscar la estación y devolver el valor correspondiente solicitado. Si no existe una estación con ese nombre, deberá mostrar el mensaje `Wrong station name` y devolver -1. Finalmente, el método `getAvgTemp` devolverá la media de las temperaturas (`T`) de todas las estaciones meteorológicas para un determinado mes (si no hay estaciones devolverá `0`), y `getWarmestStation` devolverá el nombre de la estación que tiene la temperatura máxima (`TM`) más alta para el mes indicado (si no hay estaciones devolverá la cadena vacía).

Dado el siguiente fichero `main.cc`, que puedes compilar con `g++ Station.cc Region.cc main.cc -o ej2`:

```
#include "Region.h"

int main() {
    Region r("Comunitat Valenciana");

    r.addStation("Alicante.txt");
    r.addStation("Valencia.txt");
    r.addStation("Castellon.txt");
    r.addStation("unknown.txt");  // No existe

    cout << r.getValue("Alicante",Tm,Sep) << endl;
    cout << r.getAvgTemp(Oct) << endl;
    cout << r.getWarmestStation(Aug) << endl;
}
```

La salida debe ser:

```
Error opening file
18.5
19.4667
Alicante
```

[5] Se asume que el formato del fichero será siempre correcto y que los meses aparecerán siempre ordenados.

[6] Recuerda que los tipos enumerados se almacenan internamente como valores enteros, por lo que el ejemplo anterior sería exactamente igual que hacer `data[1][3]`, ya que `TM` se almacena internamente como un 1 (la posición que ocupa en el tipo enumerado) y `Apr` como un 3 (por la misma razón).