Curso 2024-2025
Estructura de Datos y Algoritmia
Práctica 1

Departamento de Lenguajes y Sistemas Informáticos

Condiciones de entrega

- **Plazo de entrega: desde el lunes 14 de octubre hasta el viernes 18 de octubre a las 23:59 horas.**
- Se debe entregar la práctica en un fichero comprimido con todos los ficheros de la siguiente manera:

Terminal
```
tar cvfz practica1.tgz CityInfo.h CityInfo.cc Coordinates.h \
Coordinates.cc City.h City.cc Map.h Map.cc maptest.cc
```

Normas generales

- La práctica debe ser un trabajo original de la persona que entrega; **en caso de detectarse indicios de copia de una o más entregas se suspenderá la práctica con un 0 a todos los implicados.**
- Debes entregar la práctica exclusivamente a través del servidor de prácticas del Departamento de Lenguajes y Sistemas Informáticos (DLSI): `https://pracdlsi.dlsi.ua.es`
- Cuestiones que debes tener en cuenta al hacer la entrega:
  - El usuario y la contraseña para entregar prácticas son los mismos que utilizas en UACloud.
  - Puedes entregar la práctica varias veces, pero sólo se corregirá la última entrega.
  - No se admitirán entregas por otros medios, como el correo electrónico o UACloud.
  - No se admitirán entregas fuera de plazo.
- Tu práctica debe poder ser compilada sin errores ni warnings con el compilador de C++ existente en la distribución de Linux de los laboratorios de prácticas.
- Si tu práctica no se puede compilar su calificación será 0.
- La corrección de la práctica se hará de forma automática, por lo que es imprescindible que respetes estrictamente los textos y los formatos de salida que se indican en este enunciado.
- Al comienzo de todos los ficheros fuente entregados debes incluir un comentario con tu número de documento de identidad (NIF, NIE o pasaporte, tal como aparece en UACloud) y tu nombre. Por ejemplo:

ejemplo.cc
```
// DNI 12345678X GARCIA GARCIA, JUAN MANUEL
...
```

- Los ficheros fuente deber estar adecuadamente documentados usando comentarios en el propio código, sin utilizar en ningún caso acentos ni caracteres especiales.
- El cálculo de la nota de la práctica y su relevancia en la nota final de la asignatura se detallan en las diapositivas de presentación de la asignatura.

1. Objetivo

Esta práctica nos permitirá trabajar con la entrada-salida de ficheros de texto para la lectura de documentos en C++ y poder extraer información relevante relativa a los mismos. En particular, realizaremos:

1. Implementación y uso de entrada-salida de datos a partir de ficheros.
2. Implementación de tipos de datos para el almacenamiento de la información leída desde fichero.
3. Implementación de aplicaciones concretas con las clases definidas.

2. Descripción de los ficheros de texto

Para poder implementar las estructuras de datos adecuadas necesitamos en primer lugar leer la información almacenada en ficheros de texto que siguen un formato determinado. En concreto para esta práctica se leerá un fichero de texto que recogerá toda la información relacionada con un mapa como el de la imagen:

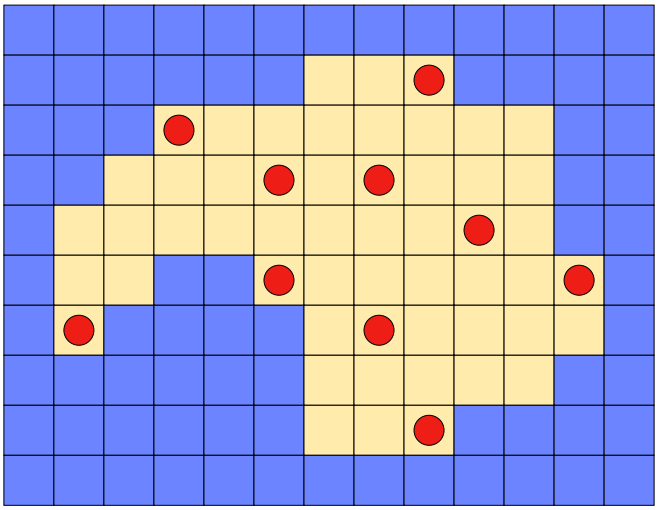

Figura 1: Mapa de una isla con varias ciudades marcadas como puntos rojos.

En primer lugar aparecerá la descripción del mapa. Cada coordenada de este mapa será un carácter indicando únicamente si es tierra `(T)` o mar `(M)`. Estos mapas serán siempre rectangulares, es decir, todas las filas tendrán la misma longitud.

Por ejemplo, el mapa de la figura anterior sería el siguiente:

```
MMMMMMMMMMMMM
MMMMMMTTTMMMM
MMMTTTTTTTTMM
MMTTTTTTTTTMM
MTTTTTTTTTTMM
MTTMMTTTTTTTM
MTMMMMTTTTTTM
MMMMMMTTTTTMM
MMMMMMTTTMMMM
MMMMMMMMMMMMM
```

Hay que tener en cuenta que este mapa sólo contendrá la información de casillas con tierra y mar. Después, se sustituirá en el mapa la letra `T` (de tierra) por la letra `C` (de ciudad) una vez leída la información de las ciudades en el resto del fichero. Las ciudades son las que aparecen con un círculo rojo en la figura anterior.

En el mismo fichero, y a continuación del mapa, aparecerá la localización e información turística de ciudades situadas en dicho mapa. En el siguiente ejemplo puedes ver un fichero completo con la información de varias ciudades después del mapa:

mapa.txt

```
MMMMMMMMMMMM
MMMMMMTTTMMMM
MMMTTTTTTTTMM
MMTTTTTTTTTMM
MTTTTTTTTTTMM
MTTMMTTTTTTTM
MTMMMMTTTTTTM
MMMMMMTTTTTMM
MMMMMMTTTMMMM
MMMMMMMMMMMM
<CIUDAD>
Auckland
1 8
<INFO>
museo 3
hotel 3
restaurante 4
aeropuerto
<CIUDAD>
Rotorua
2 3
<INFO>
museo 5
monumento 3
*** Waimangu Volcanic Valley
hotel 10
restaurante 10
<CIUDAD>
Matamata
3 5
<INFO>
museo 1
monumento 3
hotel 1
restaurante 12
** Hobbiton Movie Set
<CIUDAD>
Hamilton
3 7
<INFO>
hotel 2
restaurante 1
```

Cada ciudad empezará con una etiqueta `<CIUDAD>`, e irá seguida del nombre y las coordenadas de la ciudad en el mapa. A continuación vendrá la información relevante que vimos en la práctica anterior, precedida por la etiqueta `<INFO>`. Esta información puede estar en cualquier orden y no tienen por qué aparecer todos los elementos (aunque al menos siempre habrá uno). Algunas ciudades contienen un elemento `mostImportant` (marcado con una secuencia de `'*'`). Dependiendo de lo importante que sea ese elemento podrá tener entre 1 y 3 estrellas.

Esta información se almacenará en estructuras de datos adecuadas para su posterior procesamiento.

3. Clases

Para todas las clases descritas a continuación, puedes añadir las variables de instancia, constructores y métodos que consideres necesarios. Sin embargo, no puedes modificar los atributos y métodos que se describen aquí.

3.1. Clase CityInfo

La clase `CityInfo` especificada en la práctica 0.

3.2. Clase Coordinates

La clase `Coordinates` se definirá con:

Atributos:

- `int row;`
  Fila.
- `int col;`
  Columna.

Métodos:

- `Coordinates();`
  Constructor por defecto que inicializa a -1 los enteros para indicar que no es una coordenada válida.
- `Coordinates(int r, int c);`
  Constructor a partir de dos enteros: inicializa los enteros a los valores pasados por parámetro si son mayores o iguales a 0, y a -1 (los dos) en cualquier otro caso.
- `Coordinates(const Coordinates &c);`
  Constructor de copia.
- `Coordinates & operator=(const Coordinates &c);`
  Sobrecarga del operador de asignación.
- `int getRow() const;`
  Devuelve el valor del atributo `row`.
- `int getCol() const;`
  Devuelve el valor del atributo `col`.
- `friend ostream & operator≪(ostream &, const Coordinates &);`
  Muestra las coordenadas entre paréntesis y separadas por una coma, sin salto de línea, p.ej: `(1,8)`

3.3. Clase City

La clase `City` almacena la información de una ciudad en el mapa.

Atributos:

- `int id;`
  Identificador de la ciudad.
- `string name;`
  Nombre de la ciudad.
- `Coordinates coord;`
  Coordenadas en el mapa.

- `CityInfo info;`
  Información turística.

Métodos:

- `City(string name);`
  Constructor a partir de una cadena: los datos que no se pasan por parámetro se inicializan a sus valores por defecto, siendo -1 en el caso de la variable `id`.

- `City(const City &c);`
  Constructor de copia.

- `City & operator=(const City &c);`
  Sobrecarga del operador de asignación.

- `int setCoord(int r, int c, vector<vector<char>> &map);`
  Método que devuelve un entero y recibe por parámetro 2 enteros y una referencia a una matriz dinámica de caracteres que representa un mapa. Creará un objeto de tipo `Coordinates` con los enteros pasados como parámetro y lo almacenará en `coord`, si no tenía ya coordenadas válidas y esa posición en el mapa es una `T`. Entonces esa `T` se modifica por una `C`. Además se calcula y almacena el identificador de la ciudad de la siguiente manera: si $rows \times cols$ (filas $\times$ columnas) es el tamaño del mapa, y (r, c) las coordenadas de la localidad, el identificador se obtendrá como $(r \times cols) + c$. El método actualiza el valor de `id` si modifica el mapa devolviendo su valor. En caso de que no actualice el `id` por algún motivo devuelve -1.

- `string getName() const;`
  Devuelve el valor del atributo `name`.

- `Coordinates & getCoord();`
  Devuelve una referencia al objeto `coord` con las coordenadas de la ciudad.

- `CityInfo &getInfo();`
  Devuelve una referencia al objeto `info` con la información turística de la ciudad.

- `void setInfo(CityInfo info);`
  Método que no devuelve nada y recibe por parámetro un objeto de tipo `CityInfo`. El método debe añadir la información turística de la ciudad sustituyendo la que tuviese almacenada anteriormente.

- `int getId() const;`
  Devuelve el identificador de la ciudad.

- `friend ostream & operator≪(ostream &, const City &);`
  Muestra el identificador, guión (-), nombre de la localidad, guión (-), coordenadas según el formato de la clase `Coordinates`, nueva línea e información turística según el formato de la clase `CityInfo`.

3.4. Clase Map

La clase `Map` representa un mapa y una serie de ciudades ubicadas en ese mapa.

Atributos:

- `vector<vector<char>> map;`
  Matriz dinámica de caracteres, para almacenar el mapa leído de un fichero.

- `vector<City> cities;`
  Vector de objetos de tipo `City`, para almacenar las ciudades leídas de un fichero.

Métodos:

- `Map();`
  Constructor por defecto que inicializa las variables de instancia.

- `void read(string filename);`
  Recibe como parámetro una cadena con el nombre del fichero a leer. El método lee el fichero de texto con el formato descrito en la sección 2 (puedes asumir que no tendrá errores).

  Inicialmente se leerá la parte del fichero que corresponde a un mapa, almacenándolo en la matriz `map`. A continuación en el fichero aparecerá la información de las ciudades. Es posible que en el fichero no haya información de todos los elementos turísticos de una ciudad, por lo que se inicializarán esos campos con el valor por defecto. Las estrellas (*) se considerará que forman parte del nombre del elemento turístico `mostImportant`.

  Toda la información de las ciudades se almacenará en objetos de tipo `City` que se añadirán al final del vector `cities`.

- `vector<vector<char>> & getMap();`
  Método que no recibe ningún parámetro y devuelve una referencia al mapa de caracteres almacenado.

- `vector<City> & getCities();`
  Método que no recibe ningún parámetro y devuelve una referencia al vector de objetos de tipo `City` almacenado.

- `char getMapPosition(Coordinates coord) const;`
  Método que devuelve el carácter almacenado en la posición dada por la variable `Coordinates` pasada por parámetro. Si es una posición no válida devuelve la letra `X`.

- `friend ostream & operator≪(ostream &, const Map &);`
  Muestra el mapa como una matriz de caracteres, con el mismo formato que descrito para los ficheros pero mostrando el carácter `C` en las posiciones donde haya una ciudad. A continuación se mostrarán las ciudades del array `cities`, una por línea con el formato definido para la clase `City`.

4. Aplicación `maptest`

Implementa un fichero llamado `maptest` que recibirá como argumento de entrada el nombre de un fichero con la información de un mapa tal como se ha descrito en la sección 2.

Por ejemplo, se invocará de la siguiente forma:

Terminal

```
./maptest mapa.txt
```

El método `main` deberá seguir estos pasos:

- Cogerá el nombre del fichero de texto con la información del mapa como argumento de entrada.
- Creará un objeto de tipo `Map` y leerá la información del fichero.
- Mostrará la información del mapa siguiendo el formato descrito a continuación.

La aplicación mostrará un resumen de la información leída y situada en el mapa. Se mostrará un listado de ciudades con sus coordenadas, el total de cada tipo de elementos turísticos encontrados y un listado de monumentos con la ciudad en la que se encuentran.

La información a mostrar seguirá el siguiente formato (ver ejemplo de salida en la página siguiente):

- El mapa en formato texto (la matriz de caracteres) actualizado con la posición de las ciudades.
- Línea en blanco
- CIUDADES
- Lista de ciudades en el mapa, una por línea, mostrando el nombre y sus coordenadas

- Línea en blanco
- ESTADISTICAS
- Número total de elementos turísticos de cada tipo en el mapa
- Línea en blanco
- MONUMENTOS
- Lista de monumentos de las ciudades (si los hay), uno por línea, mostrando el nombre del monumento seguido de la ciudad entre paréntesis.

Ejemplo de salida para el fichero de la sección 2

```
MMMMMMMMMMMM
MMMMMMTTCMMMM
MMMCTTTTTTMM
MMTTTCTCTTTMM
MTTTTTTTTTMM
MTTMMTTTTTTM
MTMMMMTTTTTM
MMMMMMTTTTTMM
MMMMMMTTTMMMM
MMMMMMMMMMMM

CIUDADES
Auckland (1,8)
Rotorua (2,3)
Matamata (3,5)
Hamilton (3,7)

ESTADISTICAS
Museos: 9
Monumentos: 6
Hoteles: 16
Restaurantes: 27
Aeropuertos: 1

MONUMENTOS
*** Waimangu Volcanic Valley (Rotorua)
** Hobbiton Movie Set (Matamata)
```

5. Probar la práctica

- En Moodle se publicará un corrector de la práctica con algunas pruebas (se recomienda realizar pruebas más exhaustivas).
- El corrector viene en un archivo comprimido llamado `correctorP1.tgz`. Para descomprimirlo debes copiar este archivo donde quieras realizar las pruebas de tu práctica y ejecutar:

  Terminal

  ```
  tar xzvf correctorP1.tgz
  ```

  De esta manera se extraerá una carpeta con el siguiente contenido:

  - Fichero `corrige.sh`: *shell-script* que tienes que ejecutar.
  - Carpeta `pruebas`: dentro de esta carpeta están los ficheros con las pruebas a ejecutar.

- El fichero `README.md`: fichero con información adicional sobre el funcionamiento de las prueba e instrucciones para que puedas crear y ejecutar tus propias pruebas.

- Una vez descomprimido el fichero, debes copiar tus ficheros fuente a la misma carpeta donde está el fichero `corrige.sh`.
- Sólo queda ejecutar el *shell-script*. Primero debes darle permisos de ejecución. Para ello ejecuta:

Terminal

```
chmod +x corrige.sh
./corrige.sh
```

A. Lectura de ficheros de texto en C++

Para poder trabajar con ficheros de texto, debemos incluir la biblioteca de funciones fstream:

```
// Ejemplo de lectura de fichero de texto pasado como argumento donde devuelve
// a la salida estandar el contenido del mismo con el numero de linea al principio
#include <iostream>
#include <fstream> // Biblioteca para trabajar con ficheros

using namespace std;

// En argv[0] se almacena el nombre del programa ejecutado
// En argv[1] se almacena el primer argumento pasado al programa
int main(int argc, char *argv[]){
  ifstream fichero(argv[1]); // Declaramos la variable para abrir el fichero

  if( !fichero.is_open() ){  // Comprobamos que no hay problemas con el fichero
    cerr << "ERROR en lectura de fichero" << endl;
  }
  else {
    string linea;
    int i = 1;
    while (getline(fichero, linea)) {
      // Esto es solo un ejemplo, habria que procesar la linea
      // Escribimos numero de linea y string leido
      cout << "linea " << i << ": " << linea << endl;
      i++;
    }
  }
}
}
```

Si además queremos procesar los string en los que tenemos palabras separadas por espacios en blanco debemos hacer lo siguiente cada vez que leemos una línea y la almacenamos en un string:

```
#include <iostream>
#include <sstream> // Necesario para usar stringstream
using namespace std;

int main() {
  string s = "No temo a los ordenadores, lo que temo es quedarme sin ellos";
  stringstream iss(s);
  string token;

  while(iss >> token) {
    // En token se guardan las subcadenas separadas por espacios en blanco
    cout << "Substring: " << token << endl;
  }
}
```