Curso 2019-2020
Estructura de Datos y Algoritmia
Práctica 1

Departamento de Lenguajes y Sistemas Informáticos

Objetivo

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

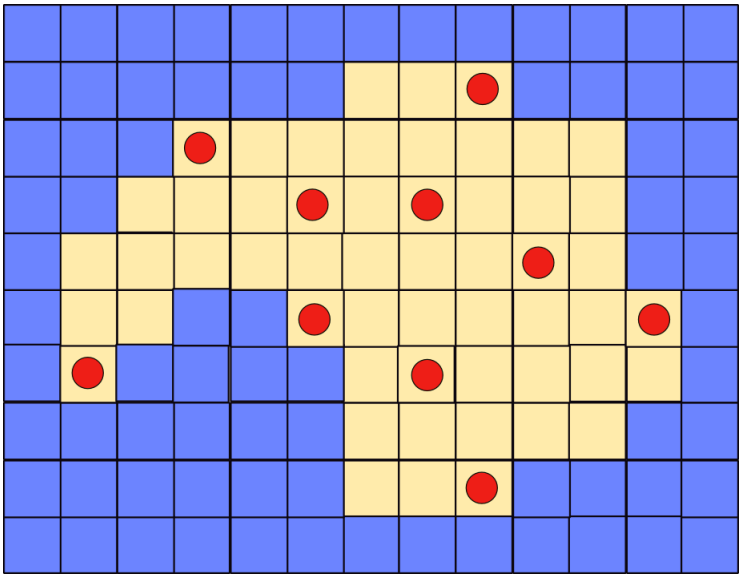
1. Implementación y uso de entrada-salida de datos a partir de ficheros en C++ y Java.
2. Implementación de tipos de datos para el almacenamiento de la información leída desde fichero en C++ y Java.
3. Implementación de aplicaciones concretas con las clases definidas.

Fechas importantes

- Plazo de entrega: desde el 21 de octubre hasta el **25 de octubre a las 23:59 horas** en el servidor de prácticas del DLSI.

1. Manipulación de 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 una región del globo terráqueo:



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 anteriormente indicada. Después, se sustituirá en el mapa la letra `T` (de tierra) por la letra `L` (de localidad), una vez leída esta información en el resto del fichero. Las localidades son las que aparecen con un círculo rojo en la figura anterior.

A continuación en el fichero aparecerá la localización e información turística de localidades situadas en dicho mapa;
Un ejemplo de sección del fichero con el formato para almacenar las localidades sería el siguiente:

```
<LOCALIDAD>
Bhosa
1 8
<INFO>
museo 3
hotel 3
restaurante 4
<LOCALIDAD>
Sinrola
2 3
<INFO>
museo 5
monumento 3
** la fontana di trevi
hotel 10
restaurante 10
<LOCALIDAD>
Calari
3 5
<INFO>
museo 1
monumento 3
hotel 1
restaurante 12
* la playa de la caleta
aeropuerto
<LOCALIDAD>
Asti
3 7
<INFO>
hotel 2
restaurante 1
<LOCALIDAD>
...
```

Como se puede observar en el ejemplo, la información relevante de cada localidad puede estar en cualquier orden, no tienen por qué tener todos los ítems (aunque al menos tendrán uno), y algunos de ellos contienen un top (marcado con una cadena de `'*'`). Dependiendo de lo importante de ese top podrá tener entre 1 y 3 estrellas.

La información almacenada en el fichero se leerá automáticamente y se almacenará en estructuras de datos adecuadas para su posterior procesamiento.

2. Clases

2.1. Clase InfoTur

La implementación de esta clase en C++ es la que habéis realizado en la práctica 0. Para la implementación en Java se necesita, además de las mismas variables de instancia, sólo los siguientes constructores y métodos:

- `public InfoTur()`: constructor por defecto;
- `public InfoTur(int i, int j, int k, int l, boolean b )`: constructor con parámetros;
- `public ArrayList<Integer> getInfoTur();`
- `public String getMasFrecuente();`
- `public void setTop( String n );`
- `public String getTop();`
- `public String toString();`

2.2. Clase Coordenadas

La clase `Coordenadas` se definirá con:

- las siguientes variables de instancia:
  - un entero: indica la fila
  - un entero: indica la columna
- y los siguientes métodos:
  - constructor por defecto que inicializa a -1 los enteros para indicar que no es una coordenada válida;
  - 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 en cualquier otro caso;
  - `getFila`: método que no tiene paso de parámetro y devuelve la fila;
  - `getColumna`: método que no tiene paso de parámetro y devuelve la columna;

2.3. Clase Localidad

La clase `Localidad` que hay que implementar contendrá:

- las siguientes variables de instancia:
  - `nombre`: una cadena que indica el nombre de la localidad;
  - `coor`: una variable de tipo `Coordenadas` que indica su posición en el mapa;
  - `info`: una variable de tipo `InfoTur` con la información turística;
  - `id`: una variable de tipo entera para indicar el identificador de la localidad;
- y los siguientes métodos de instancia:
  - 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`;
  - `setCoor`: método que devuelve un entero y recibe por parámetro 2 enteros y una referencia a una matriz dinámica de caracteres que representa un mapa. Se crea un objeto de tipo `Coordenadas` con los enteros pasados como parámetro y lo almacena en `coor`, si no tenía ya coordenada válida y esa posición en el mapa es una `T`. Entonces esa `T` se modifica por una `L`. Además se calcula y almacena el identificador de la localidad de la siguiente manera: si $m \times n$ (filas $\times$ columnas) es el tamaño del mapa, y (x, y) las coordenadas de la localidad, el identificador se obtendrá como $n \times x + y$. 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.
  - `setInfo`: método que no devuelve nada y recibe por parámetro un objeto de tipo `InfoTur`. El método debe añadir la información turística de la ciudad sustituyendo la que tuviese almacenada anteriormente;
  - `getNombre`: método sin paso de parámetro que devuelve una cadena con el nombre de la localidad;
  - `getCoor`: método sin paso de parámetro que devuelve una referencia al objeto de tipo `Coordenadas` de la localidad;
  - `getInfo`: método sin paso de parámetro que devuelve una referencia al objeto con la información turística de la localidad;
  - `getId`: método sin paso de parámetro que devuelve el identificador de la localidad.

2.4. Clase Coleccion

La clase `Coleccion` que hay que implementar contendrá:

- las siguientes variables de instancia:

- `mapa`: matriz dinámica de caracteres, para almacenar los mapas leídos desde fichero;
- `localidades`: array dinámico de objetos de tipo `Localidad`, para almacenar las localidades leídas desde fichero;

- los siguientes métodos:
  - constructor por defecto que inicializa las variables de instancia;
  - `lectura`: método que no devuelve nada y recibe como paso de parámetro una cadena que corresponde al nombre de un fichero. El método lee el fichero de texto con el formato descrito en la sección 1 (sin errores).

    Inicialmente se leerá la parte del fichero que corresponde a un mapa almacenándolo en la matriz `mapa`. A continuación en el fichero aparecerá la información de las localidades:
    - cada localidad empieza con la cadena literal `<LOCALIDAD>`;
    - se lee el nombre de la ciudad;
    - se leen las coordenadas (dos enteros). A su vez, se almacenará en el `mapa` en dichas coordenadas, si son correctas, la letra `L`, indicando que allí hay una localidad;
    - a continuación empieza la información turística (después de la cadena literal `<INFO>`) que se almacena en un objeto de tipo `InfoTur`. Es posible que en el fichero no haya información de todos los elementos, por lo que se inicializarán esos campos con el valor por defecto;
    - toda esta información se almacenará en un objeto de tipo `Localidad` que habrá que añadir al final de su array `localidades`.
  - `getMapa`: método que no recibe ningún parámetro y devuelve una referencia al mapa dinámico de caracteres almacenado.
  - `getLocalidades`: método que no recibe ningún parámetro y devuelve una referencia al array dinámico de objetos de tipo `Localidad` almacenado.
  - `getCoorMapa`: método que devuelve el carácter almacenado en la posición dada por la variable `Coordenadas` pasada por parámetro. Si es una posición no válida devuelve la letra `X`.

NOTA: Se pueden añadir en todas las clases las variables de instancia, constructores y métodos que se consideren necesarios.

3. Lenguajes de programación

Todas las clases se implementarán en Java y C++, las particularidades a tener en cuenta en cada uno de los lenguajes son las siguientes:

3.1. Java

- Para cada una de las clases se implementará un fichero con el mismo nombre de la clase y extensión `.java`;
- Para cada una de las clases se sobreescribe el método `toString()` que permitirá visualizar los elementos de cada clase;
- los arrays dinámicos a usar serán los `ArrayList` de `java.util`;
- las cadenas se implementarán usando la clase `String`;

3.2. C++

- Para cada una de las clases se implementarán dos ficheros con el mismo nombre de la clase y extensiones `.h` y `.cc`;
- Para cada una de las clases se implementarán todas las operaciones canónicas [1];
- el destructor debe dejar el objeto en la misma situación que si hubiera sido creado con el constructor por defecto;
- los arrays dinámicos a usar serán los `vector` de la `Standard Template Library (STL)`;
- las cadenas se implementarán usando la clase `string`;

3.3. Visualización de las clases

La visualización de las clases a implementar (`toString()` en Java, y `operator<<` en C++), tendrá el siguiente formato:

- Clase `InfoTur`: el mismo formato que el definido en la práctica 0;
- Clase `Coordenadas`: paréntesis de apertura '(' seguido de primera coordenada, una coma, segunda coordenada, paréntesis de cierre ')';
- Clase `Localidad`: identificador, guión (-), nombre de la localidad, guión (-), coordenadas según el formato anterior, nueva línea e información turística según el formato del operador $<<$ (en C++) o `toString()` (en Java) definido en la clase `InfoTur`;
- Clase `Coleccion`: el mapa de la matriz de caracteres con el mismo formato que se ha leído (ver ejemplo del enunciado en la página 2). A contiunuación se mostrará el array `localidades`, una por línea con el formato definido para la clase `Localidad`;

[1] constructor de copia, destructor, operador =, operador $<<$

4. Aplicación

Implementa una clase denominada `TurRelevante` que recibirá como parámetros de entrada un argumento que representa la información del mapa y la información de las localidades en el mapa.

En la clase `TurRelevante` se ejecutará la aplicación, que consiste en mostrar la información más importante de cada localidad correctamente leída y situada en el mapa. En concreto, se trata de mostrar el nombre de la localidad con sus coordenadas, además del elemento turístico más frecuente en dicha localidad (monumento, hotel, restaurante ...) y, si lo tiene, el elemento más relevante (top). Si se produce un empate con dos ítems más frecuentes, se imprimirá el primero en orden lexicográfico.

Se implementará un método `main` que:

- detectará el parámetro de la aplicación;
- abrirá el fichero de texto con el mapa y las localidades y lo almacenará todo utilizando los tipos de datos implementados;
- mostrará por pantalla todas las localidades correctas, una por línea, según se explica en el párrafo siguiente.

La información a mostrar seguirá el siguiente formato:

- en primer lugar se mostrará el nombre de la localidad y las coordenadas de su posición en el mapa separados por un espacio en blanco y con paréntesis y coma en las coordenadas;
- a continuación un ':' y un espacio en blanco, y separados por un espacio en blanco y en este orden, el elemento más frecuente en la ciudad y el más relevante (top), ambos separados por un guión.

  Por ejemplo, se invocará de la siguiente forma:

  - en C++ → `TurRelevante fichero.txt`
  - en Java → `java TurRelevante fichero.txt`

Ejemplo de salida para los ficheros del enunciado:
Bhosa (1,8): restaurante
Sinrola (2,3): hotel - ** la fontana di trevi
Calari (3,5): restaurante - * la playa de la caleta
Asti (3,7): hotel
...

5. Normas generales

Entrega de la práctica:

- Lugar de entrega: servidor de prácticas del DLSI, dirección `http://pracdlsi.dlsi.ua.es`;
- **Plazo de entrega:** desde el lunes 21 de octubre hasta el **viernes 25 de octubre a las 23:59 horas**;
- Se deben entregar las prácticas en **dos ficheros comprimidos** (uno para el código en Java y otro para el código en C++), con todos los ficheros creados y ningún directorio de la siguiente manera:
  - `tar cvfz practica1C.tgz InfoTur.h InfoTur.cc Coordenadas.h Coordenadas.cc Localidad.h Localidad.cc Coleccion.h Coleccion.cc TurRelevante.cc`
  - `tar cvfz practica1J.tgz InfoTur.java Coordenadas.java Localidad.java Coleccion.java TurRelevante.java`
- No se admitirán entregas de prácticas por otros medios que no sean a través del servidor de prácticas;
- El usuario y contraseña para entregar prácticas es el mismo que se utiliza en UACloud;
- La práctica se puede entregar varias veces, pero sólo se corregirá la última entregada;
- Los programas deben poder ser compilados sin errores ni warnings con los compiladores de C++ y Java existente en la distribución de Linux de los laboratorios de prácticas; si la práctica no se puede compilar su calificación será 0. Se recomienda compilar y probar la práctica con el autocorrector inmediatamente antes de entregarla;
- Los ficheros fuente deber estar adecuadamente documentados usando comentarios en el propio código, sin utilizar en ningún caso acentos ni caracteres especiales;
- Es imprescindible que se respeten estrictamente los formatos de salida indicados, ya que la corrección principal se realizará de forma automática;
- Al principio de cada fichero entregado (primera línea) debe aparecer el DNI y nombre del autor de la práctica (dentro de un comentario) tal como aparece en UACloud (pero sin acentos ni caracteres especiales);

  Ejemplo:
  `DNI 23433224 MUÑOZ PICÓ, ANDRÉS` $\Rightarrow$ NO
  `DNI 23433224 MUNOZ PICO, ANDRES` $\Rightarrow$ SI

Sobre la evaluación en general:

- 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;
- La influencia de la nota de esta práctica sobre la nota final de la asignatura está publicada en la ficha oficial de la asignatura (apartado evaluación).

6. Probar la práctica

- En UACloud se publicará un corrector de la práctica para cada lenguaje de programación con algunas pruebas (se recomienda realizar pruebas más exhaustivas).
- El corrector viene en un archivo comprimido llamado `correctorP1*.tgz`. Para descomprimirlo se debe copiar este archivo donde queramos realizar las pruebas de nuestra práctica y ejecutar: `tar xfvz correctorP1*.tgz`.

  De esta manera se extraerán de este archivo:

  - El fichero `corrige.sh`: *shell-script* que tenéis que ejecutar.
  - El directorio `practica1-prueba`: dentro de este directorio están los ficheros
    - `p01.*`: programa fuente en C++ o java con un método main que realiza una serie de pruebas sobre la práctica.
    - `p01.txt`: fichero de texto con la salida correcta a las pruebas realizadas en `p01.*`.
- Una vez que tenemos esto, debemos copiar nuestros ficheros fuente (TODOS) al mismo directorio donde está el fichero `corrige.sh`.
- Sólo nos queda ejecutar el *shell-script*. Primero debemos darle permisos de ejecución. Para ello ejecutar:

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

Apéndice 1. Lectura de ficheros de texto en C++

Para poder trabajar con ficheros de texto, debemos incluir la biblioteca de funciones fstream: `#include <fstream>`

```
//ejemplo de lectura de fichero de texto pasado como argumento donde devuelve
//a la salida estándar el contenido del mismo con el número de línea al principio
#include <iostream>
#include <fstream>

using namespace std;

//en argv[0] se almacena el nombre del programa
int main(int argc, char *argv[]){

  string linea;
  ifstream fichero(argv[1]); //declaramos la variable del fichero

  if( !fichero )  //comprobamos que no hay problemas con el fichero
  {
    cerr << "ERROR en lectura de fichero" << endl;
    exit(-1);
  }

  string s;
  int i=1;
  while (getline(fichero,s))
  {
    //esto es solo un ejemplo de qué hacer con s, hay que procesarlo de todas formas
    cout << "linea " << i << ": " << s << endl; //escribimos numero de linea y string
        leido
    i++;
  }

  return(0);
}
```

Apéndice 2. Clase string de C++

Es de tamaño variable, no estando limitado el número de caracteres que contiene. Los métodos más utilizados son:

- Longitud del `string`:

```
unsigned int length();
```

- Búsqueda de subcadena:

```
// Si no se encuentra devuelve la constante string::npos
unsigned int find (const string str, unsigned int pos=0);
```

- Operaciones de `string`:
  - comparaciones ($==, !=, >, <, >=, <=$)
  - asignación ($=$)

- concatenación (+)
- acceso a componentes como si fuera un array de caracteres (sólo si el string contiene información) ([ ])

- conversión de array de caracteres a `string` (con la asignación =)
- conversión de `string` a array de caracteres (función `c_str()`)
- conversión de `int` a `string`

```
#include <sstream>
int n=100;
stringstream ss;
ss << n;
// Tambien se pueden concatenar mas cosas, por ejemplo:
// ss << "El numero es: " << n << endl;
string numero=ss.str();
```

- conversión de `string` a `int`

```
string numero="100";
int n=atoi(numero.c_str());
```

- tokenizar un string (usando la clase `stringstream`)

```
#include <iostream>
#include <sstream>

int main()
{
  stringstream iss(``ser o no ser'');
  string sal;

  while(iss>>sal)
  {
    cout << "Substring: " << sal << endl;
  }

  return(0);
}
```

Apéndice 3. Lectura de ficheros de texto en Java (recordatorio)

La lectura de un fichero de texto en Java, tal y como se vio en la asignatura de Programación II, se puede realizar de la siguiente manera:

Abrir el fichero utilizando la clase FileReader y después leerlo utilizando la clase BufferedReader para leer por líneas.

Por ejemplo:

```
import java.io.*;
// estas serian las variables de instancia
FileReader fichero=null;
BufferedReader lectura=null;

try{
  fichero=new FileReader(nombre\_fichero); //argumento pasado por parametro al metodo
      donde se lee
  lectura=new BufferedReader(fichero);

  String linea = lectura.readLine();
  // bucle lectura hasta final de fichero
  while(linea!=null) {
    linea=lectura.readLine(); //falta procesar la linea
  }
}catch(IOException e){
  System.err.println("Error con nombre\_fichero");
  System.exit(0);
}
// esto podria ir al metodo cerrar
try{
  if (fichero!=null)
    fichero.close();
  if (lectura!=null)
    lectura.close();
}catch(IOException ex){
   System.out.println(ex);
}
```

Segmentación de una cadena en distintos elementos

Una vez tenemos una línea del fichero en una variable de tipo `String`, en los ejemplos la variable `linea`, hay que segmentarla para obtener de ella las distintas palabras del texto para construir un objeto de tipo `InfoTur`.

La segmentación la podemos realizar utilizando el método `split` de la clase `String` de Java que ya se vió en Programación II:

- Este método devuelve un array de `String` a partir de uno dado usando el separador que se especifique. Por ejemplo, supongamos que estamos leyendo el una información turística y tenemos la variable `linea` de tipo `String` que contiene la cadena

```
hotel 5
```

de la cual queremos obtener por separado los diferentes elementos que contiene.

Para ello primero debemos indicar el separador y después segmentar usando el método `split`.

Al método `split` le podemos indicar que el separador pueden ser distintos caracteres utilizando una expresión regular, que indicamos de la siguiente manera:

```
//indicamos el separador de campos
 String separador = "#";
```

```
//segmentamos
 String[] elems = linea.split(separador);
// En el caso que elems sea el ejemplo anterior ...
// elems[0] contendrÃa el string hotel
// elems[1] contendrÃa el string 5
```

- en el caso particular de esta práctica,

```
  separador="[ ]+"
```