# Práctica 2: Megabot 3000 (parte 2)

## Programación 2

Curso 2019-2020

Esta práctica consiste en la ampliación de la Práctica 1, añadiendo las opciones de importar/exportar ficheros de texto, guardar y recuperar datos de un fichero binario, pasar argumentos al programa por línea de comando y configurar el *chatbot*. Además se añadirá una nueva medida de similitud entre cadenas. Los conceptos necesarios para desarrollar esta práctica se trabajan en el *Tema 1*, *Tema 2* y *Tema 3* de teoría.

## Condiciones de entrega

- La fecha límite de entrega para esta práctica es el **viernes 10 de abril**, hasta las **23:59**
- Debes entregar un único fichero llamado `prac2.cc` con el código de todas las funciones

## Código de honor

Si se detecta copia (total o parcial) en tu práctica, tendrás un **0** en la entrega y se informará a la dirección de la Escuela Politécnica Superior para que adopte medidas disciplinarias

Está bien discutir con tus compañeros posibles soluciones a las prácticas
Está bien apuntarte a una academia si sirve para obligarte a estudiar y hacer las prácticas

Está mal copiar código de otros compañeros para resolver tus problemas
Está mal apuntarte a una academia para que te hagan las prácticas

Si necesitas ayuda acude a tu profesor/a
No copies

## Normas generales

- Debes entregar la práctica exclusivamente a través del servidor de prácticas del Departamento de Lenguajes y Sistemas Informáticos (DLSI). Se puede acceder a él de dos maneras:
  - Página principal del DLSI (`https://www.dlsi.ua.es`), opción "ENTREGA DE PRÁCTICAS"
  - Directamente en la dirección `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 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
- El 70 % de la nota de la práctica dependerá de la corrección automática, por lo que es imprescindible que respetes estrictamente los textos y los formatos de salida que se indican en este enunciado. El otro 30 % dependerá de la revisión manual del código que haga tu profesor de prácticas, por lo que debes también ajustarte a la guía de estilo de la asignatura
- Al comienzo de todos los ficheros fuente entregados debes incluir un comentario con tu NIF (o equivalente) y tu nombre. Por ejemplo:

prac2.cc

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

- El cálculo de la nota de la práctica y su relevancia en la nota final de la asignatura se detallan en las transparencias de presentación de la asignatura (*Tema 0*)

# 1. Descripción de la práctica

La verdadera amistad, la que se forja en la cueva de la EPS, es para toda la vida. Finalmente te has reconciliado con tus compañeros y ahora quieres que todos conozcan y disfruten de *Megabot 3000*. Para que todo el mundo pueda contribuir y disfrutar de la base de conocimiento de tu robot conversacional, en esta práctica deberás de añadir, a las funcionalidades ya implementadas en la Práctica 1, el código necesario para importar y exportar datos de intenciones usando ficheros de texto, almacenar y recuperar todos los datos del *chatbot* mediante ficheros binarios y manejar argumentos del programa por línea de comando. Además, se añadirá una nueva opción al menú principal para configurar el programa, así como una nueva medida de similitud para comparar cadenas de texto.

# 2. Detalles de implementación

En la web de la asignatura se publicarán varios ficheros que necesitarás para la correcta realización de la práctica:

- `newErrorFunction.cc`. Este fichero contiene una versión actualizada del tipo enumerado `Error`, con nuevos errores que se pueden dar en esta Práctica 2, como por ejemplo `ERR_FILE`. También contiene una actualización de la función `error`, con los mensajes correspondientes a los nuevos tipos de error introducidos. Copia el contenido de este fichero en tu código de la Práctica 1 sustituyendo al tipo enumerado `Error` y a la función `error` que tenías antes
- `converter.o`. Fichero en código objeto (compilado para máquinas Linux de 64 bits) que contiene una función `cleanString` que recibe como parámetro una cadena de texto y devuelve esa misma cadena sin acentos, ni diéresis, ni eñes (consulta la Práctica 1, Sección 6.1.3, para más información)
- `autocorrector-prac2.tgz`. Contiene los ficheros del autocorrector para probar la práctica con algunas pruebas de entrada. La corrección automática de la práctica se realizará con un corrector similar, con estas pruebas y otras más definidas por el profesorado de la asignatura
- `prac2`. Fichero ejecutable de la práctica (compilado para máquinas Linux de 64 bits) desarrollado por el profesorado de la asignatura, para que puedas probarlo con las entradas que quieras y ver la salida correcta esperada

- `intentions.txt`. Fichero de texto con algunas intenciones de ejemplo para poder ser importadas en el *chatbot*, siguiendo el formato y el procedimiento descrito en la Sección 5.2

A diferencia de la Práctica 1, para cada una de las nuevas opciones que se van a añadir al menú del programa deberás decidir tú los nombres de las funciones que vas a utilizar, sus parámetros y los valores que devuelven.

# 3. Componentes del programa

Además de los registros que se definieron en la Práctica 1 (`Example`, `Intent` y `Chatbot`), se deben definir dos nuevos registros para poder leer y escribir información sobre frases de ejemplo, intenciones y *chatbots* en ficheros binarios. Esto es necesario porque los registros mencionados contienen campos de tipo `string` y/o `vector` que, como sabes, no pueden utilizarse para leer o escribir información de ficheros binarios al no tener un tamaño fijo (si no sabes de que estamos hablando, consulta la transparencia 33 del *Tema 3* de teoría).

Los dos registros que se van a definir a continuación, `BinIntent` y `BinChatbot`, sólo se utilizarán cuando haya que leer o escribir en un fichero binario. Para el resto de funcionalidades de la práctica, se seguirán usando los registros `Example`, `Intent` y `Chatbot`, como se hacía en la Práctica 1.

## 3.1. BinIntent

Este tipo de registro se utilizará para leer y escribir información de una intención en el fichero binario:

```
struct BinIntent{
  char name[KMAXTEXTS];
  unsigned numExamples;
  char response[KMAXTEXTL];
};
```

Los cambios que se han introducido con respecto a `Intent` son los siguientes:

- El campo `string name` se sustituye por el campo `char name[KMAXTEXTS]`, donde `KMAXTEXTS` es una constante de tipo entero cuyo valor será `15`
- Aparece un campo `unsigned numExamples` que indica el número de frases de ejemplo que contiene el `Intent`
- El campo `string response` se sustituye por el campo `char response[KMAXTEXTL]`, donde `KMAXTEXTL` es una constante de tipo entero cuyo valor será `1000`

## 3.2. BinChatbot

Este tipo de registro se utilizará para leer y escribir información de un *chatbot* en el fichero binario:

```
struct BinChatbot{
  float threshold;
  char similarity[3];
  unsigned numIntents;
};
```

Los cambios que se han introducido con respecto a `Chatbot` son los siguientes:

- El campo `int nextId` desaparece, ya que no se almacenará en fichero
- Aparece un campo `unsigned numIntents` que indica el número de intenciones que contiene el *chatbot*
- El campo `vector<Intent>intents` desaparece también

# 4. Menú

El menú de la Práctica 1 se modificará añadiendo cinco nuevas opciones (`4`, `5`, `6`, `7` y `8`):

Terminal

```
1- Train
2- Test
3- Report
4- Configure
5- Import data
6- Export data
7- Load chatbot
8- Save chatbot
q- Quit
Option:
```

El funcionamiento del menú será igual que en la Práctica 1, emitiendo el error `ERR_OPTION` si la opción introducida por el usuario no coincide con ninguna de las del menú.

# 5. Opciones

En los siguientes apartados se describe el funcionamiento que deberá tener cada una de las opciones añadidas en esta práctica. Las opciones de la `1` a la `3`, así como `q`, no varían con respecto a la Práctica 1.

## 5.1. Configure

Esta opción permite configurar el *chatbot*, indicando los valores para el umbral y el algoritmo de similitud. Se activa cuando el usuario elige la opción `4` del menú principal. En primer lugar, se pedirá al usuario que introduzca el umbral de similitud que determina cuándo el sistema es capaz de contestar a una consulta o no:

Terminal

```
Enter threshold:
```

Se deberá comprobar que el valor introducido esté comprendido entre `0` y `1` (incluidos). En caso contrario, se mostrará el error `ERR_THRESHOLD`. Si el valor es correcto, se asignará al campo `threshold` del *chatbot*.

A continuación, tanto si se ha producido error como si no, se solicitará al usuario el nombre del algoritmo de similitud a utilizar con el siguiente mensaje:

Terminal

```
Enter algorithm:
```

Los valores posibles serán `jc` (Jaccard) o `ng` (n-gramas). Si se introduce cualquier otro valor, se mostrará el error `ERR_SIMILARITY`. Si es correcto, se asignará el nuevo valor al campo `similarity` del *chatbot*. Finalmente se volverá al menú principal.

## 5.2. Import data

Esta opción permite importar información sobre intenciones (nombre, frases de ejemplo y respuesta asociada) almacenada en un fichero de texto. Se activa cuando el usuario elige la opción `5` del menú principal. En primer lugar, se pedirá al usuario el nombre del fichero donde está almacenada la información, mostrando el siguiente mensaje:

Terminal

```
Enter filename:
```

A continuación, se abrirá el fichero de texto y se leerá, añadiendo al *chatbot* las intenciones almacenados en éste. Si no se puede abrir el fichero, se emitirá el error `ERR_FILE` y se volverá al menú principal.

En un fichero puede haber datos de varias intenciones. Cada una de ellas tendrá el siguiente formato:

```
#<name>#<response>
<example 1>
<example 2>
<example 3>
...
<example N>
```

Por ejemplo, el siguiente fichero muestra dos intenciones, llamadas `vacaciones navidad` (con tres frases de ejemplo asociadas) y `cordial` (con dos):

data.txt

```
#vacaciones navidad#El 24 de diciembre
¿Cuándo empiezan las vacaciones de Navidad?
Quiero saber cuándo paramos en Navidad
Dime la fecha de inicio de las vacaciones navideñas
#cordial#No tan bien como tú
¿Qué tal estás, Megabot?
¿Estás bien?
```

Las intenciones que se lean del fichero se irán añadiendo al final del vector `intents` del registro `Chatbot`, en el mismo orden en el que aparezcan en el fichero. De manera equivalente a como se hace con la opción `Add intent` del menú de entrenamiento, si al importar una intención ya existía otra con ese nombre, se emitirá el error `ERR_INTENT`, se ignorarán las frases de ejemplo asociadas y se continuará procesando el fichero por la siguiente intención (si la hubiera).



- El formato de las líneas del fichero será siempre correcto
- El fichero de entrada puede estar vacío. En ese caso, no se importará nada, pero tampoco se mostrará ningún tipo de error
- Para cada frase de ejemplo se tendrá que proceder como en la opción `Add example` del menú de entrenamiento: el texto leído se almacenará en el campo `text` de `Example` y se almacenarán en el campo `tokens` todas las palabras después de ser procesadas (que sólo contengan alfanuméricos, en minúsculas, etc.)
- En el fichero no existe información sobre el `id` de las frases de ejemplo. Para cada una de ellas se tiene que asignar de manera automática un nuevo identificador, como si la frase se hubiera creado con la opción `Add example` del menú de entrenamiento

## 5.3. Export data

Esta funcionalidad, que se activa con la opción `6` del menú principal, guardará en un fichero de texto información sobre las intenciones almacenadas en el *chatbot*: nombre, frases de ejemplo y respuesta asociadas. El formato en el que se guardará la información será el mismo que se ha descrito en la sección anterior para la importación de datos.

En primer lugar, se le preguntará al usuario si quiere guardar en el fichero todas las intenciones del *chatbot* o una en concreto, mostrando el siguiente mensaje:

```
Terminal
Save all intents [Y/N]?:
```

Si el usuario contestara con algo diferente a `Y/y` o `N/n`, se volvería a mostrar el mensaje anterior. Si el usuario contesta `Y` o `y`, se guardarán todos los elementos del vector `intents` del *chatbot* en el fichero, tal como se describe más abajo. Si contesta `N` o `n`, se le pedirá que introduzca el nombre del *Intent* concreto que quiere guardar con el siguiente mensaje:

```
Terminal
Intent name:
```

Como en otras ocasiones, si no existe una intención con ese nombre, se emitirá el error `ERR_INTENT` y se volverá al menú principal sin hacer ninguna operación sobre el fichero.

Si todo es correcto, se pedirá al usuario el nombre del fichero donde desea almacenar la información, mostrando el siguiente mensaje:

```
Terminal
Enter filename:
```

Se abrirá el fichero de texto para escritura y se guardará en él la información correspondiente, volviendo a continuación al menú principal. Si no se puede abrir el fichero se emitirá el error `ERR_FILE` y se volverá al menú principal.

- Si el fichero de salida ya existe, se sobreescribirá todo su contenido borrando lo que hubiera antes almacenado en él

## 5.4. Load chatbot

Esta opción permite cargar en el programa todos los datos de un *chatbot* a partir de un fichero binario. Los datos que hubiera en ese momento en la estructura `Chatbot` del programa se borrarán completamente y se sustituirán por la información leída del fichero. Se activa con la opción `7` del menú.

En primer lugar, se pedirá al usuario el nombre del fichero binario donde está almacenada la información, mostrando el siguiente mensaje:

```
Terminal
Enter filename:
```

Si el fichero no se pudiera abrir, se emitirá el error `ERR_FILE`. Antes de proceder a la lectura del fichero, se solicitará confirmación al usuario mostrando el siguiente mensaje:

Terminal

```
Confirm [Y/N]?:
```

Si el usuario introduce `Y` o `y`, se procederá con el borrado de los datos actuales del *chatbot* y con la lectura de los nuevos datos del fichero. Si el usuario introduce `N` o `n`, se volverá a mostrar el menú principal sin llevar a cabo ninguna acción. En caso de que el usuario introduzca cualquier otro carácter, se volverá a mostrar el mismo mensaje solicitando la confirmación.

La información del *chatbot* almacenada en el fichero binario tendrá la siguiente estructura:

- En primer lugar aparecerá un registro de tipo `BinChatbot` con los datos del *chatbot*: umbral (`threshold`), algoritmo de similitud (`similarity`) y número de intenciones guardadas (`numIntents`)
- A continuación aparecerán tantos registros de tipo `BinIntent` como indique el valor `numIntents` del registro `BinChatbot` leído en el paso anterior. Cada registro contendrá el nombre de la intención (`name`), el número de frases de ejemplo asociadas (`numExamples`) y la respuesta correspondiente (`response`)
- Detrás de cada registro de tipo `BinIntent` aparecerá un array de caracteres de tamaño `KMAXTEXTL` por cada una de las frases de ejemplo asociadas a esa intención. Habrá tantas frases como indique el campo `numExamples` del `BinIntent` leído. A cada una de estas frases de ejemplo tendrás que asignarle un `id` (comenzando por el `1`) y obtener sus *tokens* al guardarlas en el `Intent` correspondiente, como si se hubiesen creado mediante la opción `Add example` del menú de entrenamiento

A continuación se muestra un ejemplo de cuál sería la secuencia de información almacenada en el fichero binario para un *chatbot* que contuviera la información que se mostraba en el ejemplo dado más arriba, en la opción `Import data`:

| BinChatbot | BinIntent | char[KMAXTEXTL] | BinIntent | char[KMAXTEXTL] |
|---|---|---|---|---|
| threshold | name | ¿Cuándo empiezan las vacaciones de Navidad? | name | ¿Qué tal estás, Megabot? |
| 0.5 | vacaciones navidad | Quiero saber cuándo paramos en Navidad | cordial | ¿Estás bien? |
| similarity | numExamples | Dime la fecha de inicio de las vacaciones navideñas | numExamples | |
| jc | 3 | | 2 | |
| numIntents | response | | response | |
| 2 | El 24 de diciembre | | No tan bien como tú | |

- Se asume que todos los datos almacenados en el fichero binario son correctos, por lo que no es necesario hacer ninguna comprobación adicional
- Para eliminar todos los datos de un vector STL puedes usar `clear()`. Más información en el siguiente enlace: `http://www.cplusplus.com/reference/vector/vector/clear/`

## 5.5. Save chatbot

Esta opción permite guardar todos los datos del *chatbot* en un fichero binario. Se activa con la opción `8` del menú. En primer lugar, se pedirá al usuario el nombre del fichero binario donde se almacenará la información, mostrando el siguiente mensaje:

Terminal

```
Enter filename:
```

Si el fichero no se pudiera abrir, se emitiría el error `ERR_FILE`. Si el fichero ya existe, se sobrescribirá todo su contenido borrando lo que hubiera antes almacenado en él. El formato de salida será exactamente el mismo que se ha descrito en el apartado anterior para la opción `Load chatbot`.



- Ten en cuenta que puede que tengas que recortar los campos de texto (por ejemplo, `name` y `response`) al guardarlos en el fichero binario. Por ejemplo, podrías tener un `Intent` con una cadena de 40 caracteres almacenada en el campo `name`. Sin embargo, en el fichero binario tienes que guardar una estructura de tipo `BinIntent` cuyo campo `name` tiene un tamaño máximo de `15`. En este caso, tendrías que recortar el nombre original y almacenar solo los primeros 14 caracteres en fichero (el máximo admitido, ya que tienes que dejar un espacio para incluir el '\0' final)

### 5.6. Distancia basada en n-gramas

Vamos a añadir una nueva medida de similitud entre cadenas: la distancia basada en n-gramas. Un *n-grama* es una secuencia contigua de *n* caracteres extraída a partir de una palabra. Por ejemplo, un *1-grama* representa un único carácter, un *2-grama* (bigrama) dos caracteres consecutivos, un *3-grama* (trigrama) tres caracteres consecutivos, etc.

En esta práctica, nos vamos a centrar en el uso de 3-gramas. Por ejemplo, para la palabra `programar`, la secuencia de todos los 3-gramas que contiene sería la siguiente: `pro`, `rog`, `ogr`, `gra`, `ram`, `ama` y `mar`.

Para medir la similitud entre una consulta del usuario $A$ y una frase de ejemplo $B$, en primer lugar se extraerán los 3-gramas de cada palabra (*token*) de $A$ y se eliminarán los duplicados. A continuación se hará lo mismo para $B$. Finalmente, se aplicará la similitud de Jaccard pero, en esta ocasión, en lugar de entre palabras será entre 3-gramas: se dividirá el número total de 3-gramas coincidentes entre $A$ y $B$, por el número total de 3-gramas diferentes que hay entre las dos.



- Por ejemplo, si $A$ es "hola lola", los 3-gramas extraídos serán "hol", "ola" y "lol". Aunque "ola" aparezca dos veces, se considera una única vez para el cálculo de la similitud
- Si en $A$ o $B$ aparece una palabra con menos de tres caracteres, directamente se eliminará, ya que no es posible extraer un 3-grama completo para ella
- En la función `report`, el nombre que le daremos al algoritmo al mostrarlo por pantalla será `N-grams` (el otro valor posible es `Jaccard`)

## 6. Argumentos del programa

En esta práctica, el programa debe permitir al usuario indicar algunas acciones a realizar mediante el paso de argumentos por línea de comando, utilizando para ello los parámetros `int argc` y `char *argv[]` de la función `main`. El programa admitirá los siguientes argumentos por línea de comando:

- `-i <fichero texto>`. Permite importar intenciones de un fichero de texto, cuyo nombre deberá indicarse a continuación de `-i`, comportándose igual que si el usuario seleccionara la opción `Import data` del menú principal (también a la hora de tratar los errores si alguno de los datos leídos es incorrecto). Por ejemplo, si se lee del fichero una intención cuyo nombre ya existe, se emitirá el error `ERR_INTENT`, se ignorarán las frases de ejemplo asociadas a esa intención y se seguirá procesando el resto del fichero. Si no se puede abrir el fichero, se emitirá el error `ERR_FILE`. El siguiente ejemplo importará los datos almacenados en el fichero de texto `data.txt`:

Terminal

```
$ prac2 -i data.txt
```

- `-l <fichero binario>`. Permite cargar los datos del *chatbot* desde un fichero binario cuyo nombre deberá indicarse a continuación de `-l`, comportándose igual que si el usuario seleccionara la opción `Load chatbot` del menú. En este caso no deberá pedirse confirmación al usuario para cargar los datos desde el fichero. Si no se puede abrir el fichero, se emitirá el error `ERR_FILE`. El siguiente ejemplo cargará los datos almacenados en el fichero binario `data.bin`:

```
Terminal
$ prac2 -l data.bin
```

- `-t`. Pone en marcha el programa directamente en el modo de conversación con el *chatbot*, como cuando se elige la opción `Test` del menú principal. Al igual que al usar dicha opción, el diálogo acabará cuando el usuario escriba `q`, volviendo en ese caso al menú principal del programa. En el siguiente ejemplo, se pone en marcha el programa y se muestra como el `chatbot` saluda al usuario y queda a la espera de sus consultas:

```
Terminal
$ prac2 -t
>> ¿Qué puedo hacer hoy por ti?
<<
```

- Todos estos argumentos son opcionales y pueden no aparecer en la llamada al programa
- Un determinado argumento solo puede aparecer una vez en la llamada, de lo contrario se considerará que los argumentos son erróneos
- Si al introducir la opción `-i` o `-l` se produce un error de tipo `ERR_FILE`, se mostrará el error correspondiente y se iniciará el programa normalmente mostrando el menú principal
- Los ficheros binarios no tienen por qué tener extensión `.bin` y los de texto no tienen por qué tener extensión `.txt`. Utilizamos estas extensiones en los ejemplos para que quede más claro con qué tipo de fichero estamos trabajando

En caso de que se produzca un error en los argumentos de entrada (por ejemplo, que se introduzca una opción que no existe), se deberá emitir el error `ERR_ARGS` y se finalizará el programa sin llegar a mostrar el menú principal. Si todos los argumentos son correctos, se procesarán las opciones introducidas por el usuario y posteriormente se mostrará el menú principal de programa de la forma habitual. Los argumentos pueden ser erróneos si se da alguna de las siguientes circunstancias:

- Alguna de las opciones aparece repetida. Por ejemplo:

```
Terminal
$ prac2 -l data.bin -l moredata.bin
```

- A las opciones `-l` o `-i` les falta el argumento de detrás. Por ejemplo:

Terminal

```
$ prac2 -i
```

- Aparece un argumento que no es ninguno de los permitidos. Por ejemplo:

Terminal

```
$ prac2 -n data.txt
```

Todos estos argumentos pueden aparecer al mismo tiempo en una llamada al programa y además en cualquier orden. Independientemente del orden en el que aparezcan en la llamada, el orden en el que se procesarán los argumentos será el siguiente:

1. En primer lugar se procesará la opción `-l` para cargar datos de un fichero binario
2. En segundo lugar se ejecutará la opción `-i` para importar datos desde un fichero de texto
3. Finalmente, se ejecutará la opción `-t` que pone en marcha la conversación con el *chatbot*

En el siguiente ejemplo, en primer lugar se cargarán los datos del *chatbot* almacenados en el fichero `data.bin`, a continuación se incorporarán los *intents* almacenados en el fichero `data.txt` y, finalmente, se pondrá en marcha la conversación con el *chatbot*:

Terminal

```
$ prac2 -i data.txt -t -l data.bin
```

Otro ejemplo, cuyos parámetros son válidos, sería el siguiente:

Terminal

```
$ prac2 -l -l -i -t
```

Aunque a primera vista los parámetros pudieran parecer incorrectos, en realidad no lo son. El programa recibe un primer parámetro `-l` que va seguido de un fichero que se llama `-l`. A continuación recibe un parámetro `-i` que va seguido de un fichero que se llama `-t`. Esta secuencia de parámetros es, por tanto, correcta.