1. Objetivo:

Usar un lenguaje de marcado simple basado en *Markdown* para, a través de *Lex*, convertirlo en un lenguaje de marcado tipo *XML*, como *AIML*, que se emplea, por ejemplo, en la práctica 1 de Inteligencia Artificial. Por ello, la fusión de estas la he decidido llamar "*MarkAIML*".

2. Sobre *Markdown*:

Markdown es un lenguaje de marcado ligero creado por John Gruber que trata de conseguir la máxima legibilidad y facilidad de publicación tanto en su forma de entrada como de salida, inspirándose en muchas convenciones existentes para marcar mensajes de correo electrónico usando texto plano**.** Se distribuye bajo licencia BSD y se distribuye como plugin en diferentes sistemas de gestión de contenidos**.**

3. Sobre *AIML*:

El *AIML*, o *Artificial Intelligence Mark-up Language* es un lenguaje de programación basado en *XML*. Fue diseñado específicamente para ayudar en la creación de la primera entidad chatbot informática de lenguaje artificial online o *A.L.I.C.E.*, en sus siglas en inglés de *Artificial Linguistic Internet Computer Entity Chatterbot* ((en inglés) Alice). Aunque descrito muy ampliamente, el lenguaje *AIML* está especializado en la creación de agentes software con lenguaje natural, conocidos como Alicebots.

4. Los bloques de *MarkAIML*:

Al ser un lenguaje que se marca por etiquetas como <aiml>...</aiml>, he decidido hacerlo más pequeños y simbólicos con bloques delimitados por corchetes. Estos bloques son de dos tipos:

- Bloques únicos : Aquellos que tengan una etiqueta solamente (como `<star index=1/>`) o aquellos que puedan usar toda una linea de código para expresar todo (como `<pattern>...</pattern>`). Estos solamente tienen un bloque que se pone al principio de la línea o de forma puntual .
- Bloques de abrir/cerrar : Son aquellos que abren en una linea y cierran tras varias lineas de codigo (como `<category></category>`, que tienen un pattern y un template en medio). Estas precisan de dos bloques: una que abre con su tag de apertura correspondiente, y otra que lo cierra.

5. Un archivo *MarkAIML*

Un archivo AIML debe abrirse con la version de *XML* que lleva y una etiqueta `<aiml>`. Tras la cual, se suceden una serie de estructuras que siguen este patrón:

```
<category>
      <pattern>...</pattern>
      <template>...</template>
</category>
```

Y cierra con el tag `</aiml>`.

En este caso, MarkAIML lo hace bastante parecido. Inicia con dos bloques `[XML]` y `[AIML]` seguidos de este patrón:

```
[<->]
[>>] …
[<<]  …
[]
```

Y cierra con el bloque `[END]`.

Dicho la estructura del archivo, convendría mostrar los bloques implementados, ya que se eligieron los suficientes para hacer una versión mínimamente funcional:

`"[XML]"` Se pone al principio del documento y dice que el archivo es XML
`"[AIML]"` Se pone tras el anterior e indica que comienza un archivo AIML
`"[<->]"` Abre una categoría, es decir, un comportamiento del bot que al recibir un patrón dé una respuesta
`"[]"` Cierra una categoría
`"[>>]"` Representa el patrón que se debe dar al bot para producir una respuesta.
“`[<<]`” Representa la respuesta a un patrón dado. Dentro de este se pueden introducir los bloques mencionados abajo
`"[>JS:]"` Introduce un script de JavaScript, pero al parecer, desde la versión 2 no se puede usar.
`"[->]"` Abre un topic, es decir, una serie de patrones asociados a un tema concreto (no testeado)
`"[|#|]"` Aprende una cosa concreta guardándolo en un archivo
`"[/]"` Calcula algo sin escribirlo por el programa
`"[--]"` Abre un elemento de una lista. Se usa en caso de dar respuesta aleatoria
`"[@]"` Cierra un elemento de una lista.
`"[?]"` Simboliza una lista de elementos para responder aleatoriamente.
`"[aA]"` Pasa todo a mayúsculas
`"[Aa]"` Pasa todo a minúsculas
`"["[0-9]"]"` Representa el índice de la estrella o wildcard que se ha puesto en el patrón. Empieza en 1.
`"[END]"` Se pone al final para indicar que se ha terminado el archivo AIML

6. La conversión de *MarkAIML* a *AIML*. Uso de *flex++*.

Para hacer todo esto, básicamente se ha puesto la asociación de un bloque de MarkAIML a AIML. Siendo esta su asociación:

```
"[XML]" = <xml…/>
"[AIML]" =<aiml version=”2.0”>
"[<->]" = <category>
"[]" = </category>
"[>>]" = <pattern>...</pattern>
```

```
“[<<]” = <template>...</template>
"[>JS:]" = <javascript>...</javascript>
"[→]" = <topic>
"[|#|]" = <learn>...</learn>
"[/]" = <think>...</think>
"[--]" = <li>
"[@]" = </li>
"[?]" = <random>...</random>
"[aA]" = <uppercase>...</uppercase>
"[Aa]" = <lowercase>...</lowercase>
"["[0-9]"]" =  <star index=[0-9]/>
"[END]" = </aiml>
```

Para hacer las tags usé dos funciones para abrir o cerrar una tag (cuya diferencia es poner una *“/”* al principio del tag) y una función auxiliar que procese lo que hay dentro en el caso de los bloques únicos.
Todo ello ha causado en consecuencia un montón de expresiones regulares de *Lex* (una por cada bloque) pero pocas funciones con poca complejidad.

Una linea de *Lex* consiste en el bloque seguido de unas llaves donde se llama a `abrirTag` o `cerrarTag` si es un bloque de abrir/cerrar, o a `abrirTag`, después `procesar` lo que hay tras esa etiqueta, y cerrar el tag.

Para probar que funcionaría en un entorno real se ha adjuntado una copia del intérprete de *AIML programAB*, el cual se usa en las prácticas de IA.

7. Pruebas.

```
[XML]
[AIML]

[<->]
[>>]Hola * soy *
[<<]Hola [2], soy [1]. Un gusto.
[]

[<->]
[>>]Piensa en algo
[<<] Vale [/] Pensando en algo
[]
```

```
<?xml version="1.0" encoding="UTF-8"?>

<aiml version="2.0">

<category>
<pattern>Hola * soy *</pattern>

<template>Hola <star index = "2"/>, soy <star index = "1"/>. Un gusto.</template>

</category>


<category>
<pattern>Piensa en algo</pattern>

<template> Vale <think> Pensando en algo</think></template>

</category>
```

```
Program AB 0.0.6.26 beta -- AI Foundation Reference AIML 2.1 implementation
No AIMLIF Files found.  Looking for AIML
(8[5])--SABES-->(1[5])--JAVASCRIPT-->(1[4])--<THAT>-->X(1)--*-->X(1)--<TOPIC>-->
X(1)--*-->Si <javascript> ...
(8[5])--PIENSA-->(1[6])--EN-->(1[5])--ALGO-->(1[4])--<THAT>-->X(1)--*-->X(1)--<T
OPIC>-->X(1)--*-->Vale <think> Pen...
(8[5])--ME-->(1[6])--LLAMO-->(1[5])--*-->(1[4])--<THAT>-->X(1)--*-->X(1)--<TOPIC
>-->X(1)--*-->Ok#Comma me acor...
(8[5])--HOLA-->(1[7])--*-->(1[6])--SOY-->(1[5])--*-->(1[4])--<THAT>-->X(1)--*-->
X(1)--<TOPIC>-->X(1)--*-->Hola <star index...
(8[5])--DIME-->(1[6])--ALGO-->(1[5])--ALEATORIO-->(1[4])--<THAT>-->X(1)--*-->X(1
)--<TOPIC>-->X(1)--*--><random> <li> 1 ...
(8[5])--FUNCIONA-->(1[4])--<THAT>-->X(1)--*-->X(1)--<TOPIC>-->X(1)--*-->Si...
(8[5])--MAS-->(2[5])--FLOJO-->(1[4])--<THAT>-->X(1)--*-->X(1)--<TOPIC>-->X(1)--*
--><lowercase> Vale...
(8[5])--MAS-->(2[5])--FUERTE-->(1[4])--<THAT>-->X(1)--*-->X(1)--<TOPIC>-->X(1)--
*--><uppercase> Vale...
(8[5])--PROGRAM-->(1[5])--VERSION-->(1[4])--<THAT>-->X(1)--*-->X(1)--<TOPIC>-->X
(1)--*-->Program AB 0.0.6...
Human: Hola Arcadia soy Ivan
Robot: Hola Ivan, soy Arcadia. Un gusto.
Human: Piensa en algo
Robot: Vale
```

8. Observaciones y conclusiones.

Usar *Lex* para proyectos pequeños como hacer un pseudolenguaje que asocie una *keyword* a una etiqueta u otro *keyword* de otro lenguaje requiere saber qué es lo que sueles usar más y ver si te conviene más hacer tal simplificación o, si de lo contrario, te viene mejor usar el lenguaje tal y como está. En el caso de *AIML*, se podría expandir este lenguaje, pero hay veces que podría no ser muy práctico. Véase, por ejemplo, el caso de que un patrón o una plantilla ocupen más de una línea (bien por comodidad o bien porque es tan largo que no se ve). Sin embargo, podría ser escalable (se pueden crear tantos bloques como puedas imaginar, solo hay que imaginarselo).

Hablando con compañeros, esa práctica la recuerdan como muy fea, repetitiva y pesada. Tras enseñarlo, lo ven un poco más cómodo de ver, ya que son símbolos más pequeños. Sin embargo es posible que las limitaciones que suponen podrían hacer que este proyecto no sea tan llamativo.