Trabajo Práctico
Teoría de Lenguajes
"Conversor de JSON a YAML"

1er cuatrimestre de 2018

Fecha de entrega: 5 de julio

1. Introducción

El objetivo del TP es crear un conversor de *JSON*[1] (JavaScript Object Notation) a *YAML*[2] (YAML Ain't Markup Language).

2. Lenguaje de entrada

JSON es un formato de intercambio de datos. Define dos tipos de estructuras, que se pueden anidar:

- Un *objeto* es un conjunto no ordenado de pares clave/valor. Se representa en la forma { *clave* `:` *valor* `,` *clave* `:` *valor* `,` ... }. Si bien la sintaxis de JSON no exige que las claves de cada objeto sean únicas, se deberá verificar que lo sean dado que es un requisito de YAML.
- Un *arreglo* es una secuencia ordenada de valores. Se representa como `[`*valor* `,` *valor* `,` ... `]`.

El valor puede ser a su vez un objeto, un arreglo, o un valor individual. Los valores individuales pueden ser cadenas, números, valores de verdad o `null`.

Los espacios, tabulaciones, saltos de línea, etc., que no se encuentren dentro de una cadena son totalmente ignorados.

En el sitio oficial de JSON se puede encontrar una descripción detallada de la sintaxis del lenguaje.

3. Lenguaje de salida

YAML es un lenguaje de serialización que pretende ser más amigable para la lectura humana que JSON, y es comunmente usado para archivos de configuración, entre otros usos. Tiene las mismas estructuras que JSON, que se denominan en este caso mapeos y secuencias. Los valores individuales (llamados escalares) incluyen todos los de JSON pero hay más y es extensible.

Mapeos y secuencias se pueden representar usando la misma notación que en JSON, en ese caso se dice que usan el estilo *flow*. Pero también se pueden representar con una línea por elemento, sin marcadores de comienzo y fin, en donde el nivel de indentación de la línea marca a qué estructura pertenece el elemento. Este es el estilo *block*. La indentación se debe realizar exclusivamente con espacios, sin tabulaciones.

Por ejemplo, la siguiente secuencia en estilo *flow*:

[1] https://www.json.org/
[2] http://yaml.org

```
[ { clave1: valor1, clave2: [ elem1, elem2 ], clave3: valor3 }, elem2,
[ elem1, elem2, elem3 ] ]
```

Se puede representar de la siguiente manera en estilo *block*:

```
-
  clave1: valor1
  clave2:
    - elem1
    - elem2
  clave3: valor3
- elem2
-
  - elem1
  - elem2
  - elem3
```

El conversor debe utilizar en su salida exclusivamente el estilo *block*. La única excepción es cuando en la entrada hay una secuencia o un objeto vacíos, que se representarán como `[]` y {} respectivamente.

4. Ejemplos

Para la siguiente entrada:

```
[ {"clave1": "valor1", "clave 2": [ 125, "Cadena 1" ], "- clave3": true},
"Cadena con salto\nde línea", [null, 35.6e9, {}] ]
```

Se podría producir la siguiente salida:

```
-
  clave1: valor1
  clave 2:
    - 125
    - Cadena 1
  "- clave3": true
- "Cadena con salto\nde línea"
-
  -
  - 35.6e9
  - {}
```

Notar que `"- clave3"` lleva comillas porque si no podría ser confundida con un elemento de una secuencia. Para la cadena con salto de línea se eligió también el estilo *double-quoted*. Existen otras formas más legibles de representar cadenas con saltos de línea en YAML que pueden implementar si desean.

Las siguientes no son entradas válidas:

- `[ {"clave1": "valor1", "clave2": [ 125, "Cadena 1"`
  (sintaxis inválida)
- `{"clave1": "valor1", "clave 2": [ 125, "Cadena 1" ], "clave 2": true}`
  (clave duplicada)

5. Modo de uso

El programa debe poder ser invocado por línea de comandos, recibiendo los datos JSON en su entrada standard y escribiendo el resultado en su salida standard. En caso de que la entrada no sea válida se lo debe informar en la salida de error standard. Opcionalmente, la entrada y salida podrá hacerse *también* a través de archivos.

6. Implementación

Hay dos grupos de herramientas que se pueden usar para generar los analizadores léxicos y sintácticos, y agregar las acciones requeridas para generar el documento de salida:

- Uno utiliza expresiones regulares y autómatas finitos para el análisis lexicografico, y la técnica LALR para el análisis sintáctico. Ejemplos de esto son flex y bison, que generan código C o C++, JLex y CUP, que generan código Java, Golex y Yacc que generan código Go, y ply, que genera código Python.
- El otro grupo utiliza la técnica ELL(k) tanto para el análisis léxico como para el sintáctico, generando parsers descendentes iterativos recursivos. Ejemplos son JavaCC, y ANTLR, que están escritos en Java. JavaCC puede generar código Java o C++. ANTLR puede generar Java, C++, Python y JavaScript, entre otros.

Si quieren usar otra herramienta, consúltenlo con alguno de los docentes.

7. Detalles de la entrega

Se deberá enviar el código a la dirección de e-mail tptleng@gmail.com.

La entrega debe incluir:

- Un programa que cumpla con lo solicitado
- El código fuente del programa.
- Informe conteniendo:
  - el código de la solución. Si se usaron herramientas generadoras de código, imprimir la fuente ingresada a la herramienta, no el código generado.
  - Las modificaciones a la gramática o indicaciones adicionales que hayan sido necesarias para construir el parser.
  - Descripción de cómo se implementó la solución.
  - Información y requerimientos de software para ejecutar y recompilar el tp (versiones de compiladores, herramientas, plataforma, parámetros, etc).
  - Casos de prueba con expresiones sintácticamente correctas e incorrectas, resultados obtenidos y conclusiones.