Universidad de los Andes > FACULTAD DE INGENIERÍA Y CIENCIAS APLICADAS

Estructuras de Datos y Algoritmos (EDA)
Tarea 2: Validador de Documentos HTML

Prof: José M. Saavedra Rondo
Ayudantes: Braulio Torres & Cristóbal Loyola & Francisco Jiménez

Septiembre, 2022

1. Objetivo

Entender la importancia de estructuras de datos dinámicas a través de aplicaciones comunes como la validación de documentos HTML.

2. Descripción

En esta tarea deberán implementar un validador de *tags* de código HTML. HTML es el formato estándar para documentos basados en *hyperlinks*. En un documento HTML, diferentes partes del texto están limitadas por ciertas etiquetas (*tags*). Cada *tag* HTML altera la apariencia de un documento. Así, estos *tags* deben ser emparejados para limitar su alcance en el documento. El emparejamiento se realiza a través de un tag que abre y otro que cierra. La forma de un *tag* que abre es "$< tag >$" y el correspondiente *tag* que cierra sigue la forma "$< /tag >$". Por ejemplo, un encabezado pues ser escrito en HTML usando el tag $< h1 >$, como muestra la siguiente línea:

```
<h1> Este es un Título </h1>
```

Algunos *tags* de HTML comúnmente utilizados incluyen:

- **body**: indica el cuerpo del documento.
- **h1**: indica un encabezado.
- **p**: representa un párrafo.
- **ol**: representa una lista numerada.
- **li**: representa un *item* de una lista.
- **center**: centra un texto.

Un ejemplo de un documento HTML usando algunos de los *tags* descritos se muestra en la Figura 1. Además, en la misma figura se muestra la visualización del documento.

Como pueden notar, la correcta escritura del código requiere que por cada *tag* que se abre exista un correspondiente *tag* que cierra. Además, un *tag* de cierre debe cerrar siempre el último *tag* que se abrió. Así, el siguiente fragmento representaría un error:

```
<body>
<center>
<h1> Estructuras de Datos </h1>
</center>
<p>
Estructuras de datos es una disciplina
con la
que todo profesional en computación
debe estar
muy bien familiarizado. Algunos temas
que se tratan son:
</p>
<ol>
<li>Listas Enlazadas</li>
<li>Pilas y Colas</li>
<li>Árboles y Heaps</li>
<li>Grafos</li>
</ol>
</body>
```

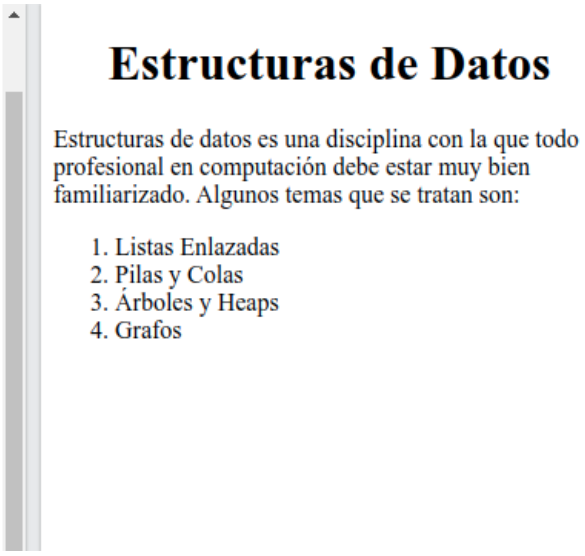

Figura 1: Izquierda: Código HTML. Derecha: Visualización del documento.

```
<h1><p> texto </h1> </p>
```

El error se debe a que estamos cerrando primero $< h1 >$ antes de $< p >$, siendo que $< p >$ fue el último en ser abierto.

Dado lo anterior, en esta tarea deberán implementar un validador de código HTML, que determine si el código tiene los *tags* correctamente emparejados o no. En el caso de determinar que el documento no es válido, deberán indicar el tipo de error y en qué línea ocurre. Dado que un documento puede tener múltiples errores de emparejamiento, es suficiente que se reporte el primer error encontrado. Por ejemplo, para el caso del código de error anterior deberán reportar algo como "Error en línea 10: Se esperaba $< /p >$ en lugar de $< /h1 >$"

3. Especificaciones

1. La entrada al programa es un texto plano con extensión html, por ejemplo *doc*1.*html*.

2. La salida del programa es un archivo de texto llamado *doc*1.*log*. Es decir, mantiene el nombre del documento, pero cambia la extensión. Este archivo reporta cada emparejamiento correcto y el primer error encontrado, si existe. En el caso de no haber errores, el archivo de salida debe contener, en la última línea, el texto "0 errores". A continuación, se presenta un ejemplo de ejecución del programa:

```
$ ./validadorHTML doc1.html

output -> doc1.log
** Contenido de doc1.log **
tag <h1> ok
tag <center> ok
tag <p> ok
tag <li> ok
tag <li> ok
tag <li> ok
tag <li> ok
tag <ol> ok
tag <body> ok
0 errores
```

3. Solamente está permitido usar las estructuras dinámicas implementadas en el curso.

4. Como mínimo deben considerar los 6 *tags* descritos anteriormente: body, h1, p, ol, li, center, pero puede agregar más si gustan.

4. Código Base

Las implementaciones deben ser propias. Solamente pueden ocupar las implementaciones que vienen en el repositorio del curso `https://github.com/jmsaavedrar/eda_cpp/`. Además, para la lectura de archivos de texto pueden mirar el ejemplo de `https://github.com/jmsaavedrar/eda_cpp/tree/main/io`.

5. Informe

1. **Abstract o Resumen**: es el resumen del trabajo.

2. **Introducción**: aquí se define el problema. (10 %)

3. **Desarrollo**: aquí se describe en detalle el diseño e implementación de los programas necesarios para realizar sus experimentos. (40 %).

4. **Resultados Experimentales y Discusión**: aquí se presentan 3 ejemplos de resultados, uno correcto y dos que reporten errores.

5. **Conclusiones**: ideas o hallazgos principales sobre el trabajo. (10 %)

6. Restricciones

1. Pueden trabajar en grupos de 2 estudiantes.

2. Todos los programas deben ser propios, permitiendo solamente utilizar el código disponible en el repositorio del curso `https://github.com/jmsaavedrar/eda_cpp/`.

3. El hallazgo de plagio será penalizado con nota 1.0, para todos los grupos involucrados.

4. Todos las implementaciones deben ser realizadas en C++. El código debe incluir un archivo README con las instrucciones de compilación y ejecución.

5. **La entrega del informe es obligatorio**. Un trabajo sin informe no será calificado, asignando la nota mínima igual a 1.0.

7. Entrega

La entrega se debe realizar por canvas hasta el domingo 02 de octubre, 2022, 23:50 hrs. La entrega debe incluir:

1. Código fuente (en C++), junto a un README con los pasos de compilación.

2. Informe