ECOTE - preliminary project

Date: **06.06.2023**

Semester: **7th**

Author and Group: **Katarzyna Paziewska, 101**

Subject: **Compiling Techniques**

I. General overview and assumptions

The task is to write an analyzer of the HTML code, that deletes any “decorations” (e.g. colors, big pictures, frames, big fonts, etc.) and to define the syntax of deleted elements.

So the input in my program is a HTML file and on the output there will be a HTML file without decorations. My program is getting rid of text styling, too big and too small pictures, fonts, breaks and italic and underlined texts. On the output there is also a file with errors.

II. Functional requirements

Restrictions about the HTML file at the input:

1. Any size of file is accepted.
2. No .css files.
3. No self closing tags.
4. No nested tags.

Grammar of a HTML file accepted:

```
htmlDocument :      '<!DOCTYPE' htmlHeader htmlBody;
htmlHeader :        'html' '>';
htmlBody :          '<head>' metadata '</head>' '<body>' content '</body>';
metadata :          title;
title :              '<title>' TEXT '</title>';
content :           (element | TEXT)*;
element :           openingTag TEXT closingTag;
openingTag :        '<' TAG_NAME attributes? '>';
closingTag :        '</' TAG_NAME '>';
attributes :        (' 'ATTRIBUTE_NAME '=' ATTRIBUTE_VALUE ‘ ‘)*;
TAG_NAME :          ‘p’ | ‘h1’ | ‘h2’ | ‘h3’ | ‘h4’ | ‘h5’ | ‘h6’ | ‘b’ | ‘u’ | ‘i’ | ‘ul’ | ‘li’ | ‘div’ | ‘img’ | ‘a’
| ‘br’ | ‘font’ ;
ATTRIBUTE_NAME : ‘href’ | ‘style’ | ‘src’ | ‘width’ |  ‘height’ | ‘alt’ | ‘ title’ | ’size’;
ATTRIBUTE_VALUE : '"' ~'"'* '"';
SCRIPT_TEXT :       ~'</script>'*;
TEXT :               ~('<' | '>' | '</' )*;
fragment LETTER :    [a-zA-Z];
fragment DIGIT :    [0-9];
```

III. Implementation

General architecture

In my design I use Python. First my algorithm is parsing the tree of HTML to check the syntax. Then it modifies certain nodes to remove the decorations and at last it makes a HTML file from this tree. So there are three parts of the algorithm.

Parts of the algorithm:

Parser

The parser checks:

1. If every start tag has to have an end tag.
2. If every closing tag matches opening tag.
3. If there is no forbidden characters in the free text.
4. Syntax of attributes.
5. Unknown attributes.
1. Unknown tags (not accepted -> error).
2. Declaration of new tags (not accepted, error).

The parser traverses through all tokens and adds appropriate nodes to a tree.

1. Create node in a tree accordingly to what a parser is currently reading:

<head>	“head” node added as a child of a root, if there is no </head> error raised. Parser reads the next word.
<title>	Title node added as a child of the root node. It checks for the next ‘<’ and if it is not </title> error is raised. Parser reads the next word.
TEXT	Free text is added as an attribute of the previous node that is a tag. If there is a character that is not allowed error is raised.
TAG_NAME	Parser checks if this name tag is in a list of possible tag names and if not error raised. Otherwise a node with a name of the tag is added as a child of head node or body node and parser checks for attributes and/or free text and checks the closing tag.
ATTRIBUTE_NAME	Parser checks if this attribute name is valid and if not error is raised. Otherwise it is added as an attribute to a current node. Next there has to be ‘=’ and ATTRIBUTE_VALUE (else error raised)

ATTRIBUTE_VALUE	Value added as an attribute of the current node node. After this parser can accept either next ATTRIBUTE_NAME or '>'

Removing decorations

At this point we have a tree of a html file. The algorithm looks for specific nodes and changes them accordingly.

1. Find all "style" nodes in text tags (not img)
   a. remove their children
   b. add new node as a child: "`style = "color:red;"`
2. Find all node with "style" attribute in img tags
   a. remove their children with height and wight
   b. add attribute "`width="100"`"
   c. add attribute "`height="100"`"
3. Find all 'u' and 'i' tags
   a. change them to 'b' tags
4. Find all 'br'
   a. remove them
5. Find all 'font' tags
   a. remove all attributes
   b. change 'font' node to 'b' node

Transforming a tree to an output HTML file

1. Traversing the tree using depth-first algorithm. This algorithm is used because the graph is constructed in a way that the correct order of elements is when you start with a root and then go as deep as possible for each branch (starting with leftmost) before backtracking.
2. For each node generating appropriate HTML elements (with attributes if they exist). It also generates closing tags for every tag. .
3. Write generated code to an output file- output.html and generate a file errors.txt with errors (may be empty).

Data structures

During the implementation I changed the data structure a little bit- attributes, their names and text are attributes of a node, not its children.

I am using a tree as a data structure to parse the HTML file and then remove decorations. This tree will have nodes that are NAME_TAGs and they have attributes:

- tagName
- children
- attributesNames
- attributesContent
- text
-

Input/output description

Input will be a HTML file and its name will be given in a function.

Output will be a new HTML file named "output.html" and an errors.txt file with errors. Examples of modification of input file to output:

1. All text tags where there is "style = … " are changed to "`style = "color:red;"`"
   Example:
   ```
   <h style="background-color: yellow;"> Example </h>
    ➪ <h style = "color:red;"> Example </h>
   ```

2. Change "style= …" in pictures so that they are all the same size.
   Example:
   ```
   <img src="img.jpg" alt="Img" width="500" height="600">
   ➪ <img src="img.jpg" alt="Img" width="100" height="100">
   ```

3. Change <u> and <i> tags to <b> tags
   ```
   <u>Example</u>
   ➪ <b>Example</b>
   ```
4. Remove all <br> tags
5. Change <font> tags to <b> tags
   ```
   <font size="10">This is some text!</font>
   ➪ <b>This is some text!</b>
   ```

Errors saved to a error.txt file:
Every error will also have information about the line where it occurred. The program will terminate on the occurrence of the first error.

Error content	Description
Error: No closing tag! Line: x	There is an opening tag that does not

	have a closing tag.
Error: Unsupported tag name “{name}”! Line: x	Tag name used is not in the list of supported names.
Error: Unsupported attribute name “{name}”! Line:x	Attribute name is not in a list of supported names.
Error: closing tag does not match opening tag! Line: x	Name in the closing tag does not match the name in the opening tag.
Error: forbidden character in a free text! Line: x	There is a character in a free text that cannot be there.
Error: Wrong syntax of an attribute! Line: x	There is no ‘=’ after ATTRIBUTE_NAME or there is no ‘ATTRIBUTE_VALUE’ in the attribute element.

IV. Functional test cases

All text cases are in a file “Functional test cases”

Here is an example:

1. HTML with multiple decorations to remove (test 15)

Input	Output

`<!DOCTYPE html>`
`  <head>`
`    <title>Test page</title>`
`  </head>`
`  <body>`
`      <div> Hello this is regular text </div>`
`      <i>Change i tag to b tag.</i>`
`      <u> Change u tag to b tag. </u>`
`    <li>list item 1 </li>`
`    <li>list item 2 </li>`
`    <li style="color:blue">list item 3 (written in blue)</li>`<br>`
</br>`<br>`   <font size="10">Text with font 10</font>`
`      <h1> Regular text </h1>`<br>`
</br>`<br>`   <h3 style="background-color:yellow"> Yellow background color</h3>`
 	
`    <h style="color:green">Company</h>`
`    <h style="color:green">Contact</h>`
	`<!DOCTYPE html>`
	`<head>`
	`<title>`
	`Test page`
	`</title>`
	`</head>`
	`<body>`
	`<div>`
	`Hello this is regular text`
	`</div>`
	`<b>`
	`Change i tag to b tag.`
	`</b>`
	`<b>`
	`Change u tag to b tag.`
	`</b>`
	`<li>`
	`list item 1`
	`</li>`
	`<li>`
	`list item 2`

```
    <h style="color:green">Country</h>
    <h>Alfreds Futterkiste</h>
    <h>Maria Anders</h>
    <h>Germany</h>
    <h>Centro comercial Moctezuma</h>
    <h>Francisco Chang</h>
    <h>Mexico</h>


  <img src="img.jpg" alt="Img" width="300"
height="200"> </img>
  </body>
</html>
```

```
</li>
<li style="color:red">
list item 3 (written in blue)
</li>
<b>
Text with font 10
</b>
<h1>
Regular text
</h1>
<h3 style="color:red">
 Yellow background color
</h3>
<h style="color:red">
Company
</h>
<h style="color:red">
Contact
</h>
<h style="color:red">
Country
</h>
<h>
Alfreds Futterkiste
</h>
<h>
Maria Anders
</h>
<h>
Germany
</h>
<h>
Centro comercial Moctezuma
</h>
<h>
Francisco Chang
</h>
<h>
Mexico
</h>
<img src="img.jpg" alt="Img" width="100"
height="100">
</img>
</body>
</!DOCTYPE html>
```

Input:

Hello this is regular text
Change i tag to b tag. Change u tag to b tag.

- list item 1
- list item 2
- list item 3 (written in blue)

Text with font 10

Regular text

Yellow background color

Company Contact Country Alfreds Futterkiste Maria Anders Germany Centro comercial Moctezuma Francisco Chang Mexico

Output:

Hello this is regular text
Change i tag to b tag. Change u tag to b tag.

- list item 1
- list item 2
- list item 3 (written in blue)

Text with font 10

Regular text

Yellow background color

Company Contact Country Alfreds Futterkiste Maria Anders Germany Centro comercial Moctezuma Francisco Chang Mexico

Result: test passed

Summary

My program passed all unit tests and functional test cases so it fulfills my assumptions and successfully removes decorations from a HTML file and parses through to check its correctness.

It also work for a file with 9000 lines of code (long_input in a folder).