Project 2: Basic HTML Parsing and Crawling

This project is divided into two parts:

1. Build a simple HTML parser that determines if your HTML tags are balanced.
2. Find the number of web pages you can visit from a certain HTML page.

To simplify this project, all pages will be local to your machine.

Part 1: Basic HTML Parsing

It is important to know if your HTML tags are balanced. For example:

```
<html>
  <body>
    <p>
      <b>Hello World!</b>
    </p>
  </body>
</html>
```

is balanced since each tag has a begin tag (`<tagname>`) followed by an end tag (`<tagname>`) at the same "level depth". For instance, consider the following:

```
<html>
  <body>
    <p>
      <b>Hello World!</b>
    </body>
  </p>
</html>
```

The above is not balanced because there is a `</body>` ending a `<p>` tag. Given the following basic HTML definition, your job is to determine if a piece of HTML is balanced or unbalanced.

HTML	`<html>BODY</html>`
BODY	`<body>TEXT</body>`
TEXT	STRING | STRING TEXT | TAG | TAG TEXT
STRING	empty or any combination of printable characters other than < and >
TAG	BOLD | DIVISION | ITALICS | LINK | PARAGRAPH
BOLD	`<b>TEXT</b>`
DIVISION	`<div>TEXT</div>`
ITALICS	`<i>TEXT</i>`
LINK	`<a href="URL">TEXT</a>`
PARAGRAPH	`<p>TEXT</p>`
URL	STRING

Part 2: Basic Web Crawler

Now that we have a basic HTML parser, write a web crawler to determine the number of unique pages that can be visited from a certain page. See the example below:

Example Output

```
./html-test pages/*
Parsing: 'pages/index.html'
Parsing: 'pages/unbalanced1.html'
Parsing: 'pages/unbalanced2.html'
Parsing: 'pages/csu.html'
Parsing: 'pages/theend.html'
                    Page  Balanced  Visit Count
        pages/index.html       ☑           2
  pages/unbalanced1.html       ☐           3
  pages/unbalanced2.html       ☐           3
          pages/csu.html       ☑           1
       pages/theend.html       ☑           0
```

As you can see in the pages directory, `csu.html`, `theend.html`, and `index.html` are balanced. Do **not** count links that are unvisitable (i.e., the page does not exist at that location). Here is the explanation of the visit amounts:

File	Visits	Explanation
`index.html`	2	Can visit `csu.html`, which is valid, and that page can visit `theend.html`.
`unbalanced1.html`	3	Can visit `index.html`, which links to `csu.html`, which links to `theend.html`.
`csu.html`	1	Can visit `theend.html`, which is valid and a dead end.
`theend.html`	0	Links to only one page, but it does not exist.

Remember, the visit count is the number of **unique** pages that can be visited from a page.

Parsing

The first part of this project requires you to read a file and parse out the input. This may seem daunting at first, but here are a couple of things to help out:

- Tags:
  - begin with a "<" and end with a ">" and are all lowercase.
  - do **not** have a < or > between the starting < and ending >. Therefore, once you see a <, parse to a >; that string is your tag and attributes.
  - All opening and closing tags, except the anchor tag (`<a>`), have no attributes or white-space. For example, the BODY tag will always be "`<body>`" with no other characters.
  - The anchor tag will **strictly** be in the form "`<a href="filename">`" with exactly one space between the "`a`" and "`href`". The URL is in double quotes after "`href=`".

- Any page with an invalid tag is unbalanced.
- To may use the extraction operator (>>) with a `char` variable to read eah character of input while ignoring whitespace (`' '`, `'\t'`, `'\n'`, `'\r'`). Alternatively, you may use the input streams `get()` method to read in each character including whitespace.

Hints

So far, we have used arrays, linked lists, pointers, stacks, and queues. Chances are you will be using multiple data structures to solve this problem.

Also, you may use `<vector>`, `<stack>`, `<queue>`, or `<deque>`, but no other STL containers.

The C++ Regular Expression Library may be used for parsing but it is optional.

Warning

The Data Structure projects are *significantly* more demanding than the labs. Start thinking about the data structure now! Please talk to me when you hit a roadblock. Before talking to me, be prepared to show me progress towards a solution. If you come with an idea, I will help guide your idea to a good solution. If you come in without any ideas, I will tell you to keep thinking about it. Keep in mind that this project has many good, fair, and terrible solutions. Design a solution that not only works but works well.

Write Test Cases

You are expected to create well written and comprehensive test cases. Your test cases will be a real consideration in your grade for this project. I highly recommend creating additional HTML files to test different conditions.

How to turn in

Turn in via GitHub. Ensure the file(s) are in your directory and then:

```
git add src/* test/* pages/*
git commit -m "Project 2 Solution"
git push
```

Assigned

October 8, 2024

Due Date

October 29, 2024, at 11:59 p.m.

Teamwork

No teamwork or generative AI is permitted; your work must be your own.