Project 2: Basic HTML Parsing and Crawling

Instructions

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 webpages you can visit from a certain HTML page.

To keep things simple for 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 given piece of HTML is balanced or not.

HTML	`<html>BODY</html>`
BODY	`<body>TEXT</body>`
TEXT	STRING | STRING TEXT | TAG | TAG TEXT
STRING	possibly empty string of printable characters other than < and >
TAG	BOLD | ITALICS | LINK | PARAGRAPH
BOLD	`<b>TEXT</b>`
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 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       yes            2
   pages/unbalanced1.html        no            3
   pages/unbalanced2.html        no            3
           pages/csu.html       yes            1
        pages/theend.html       yes            0
```

`csu.html`, `theend.html`, and `index.html` are balanced, while `unbalanced1.html` and `unbalanced2.html` are not. Also, if the link is **not** visit-able (i.e., the page does not exist), then do not count towards a possible link visit. 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 fist 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 "`>`".
  - do **not** have a `<` or `>` between the starting `<` and ending `>`. Therefore once you see a `<` parse to a `>` and that string is your tag and attributes.
  - All tags, except the anchor tag (`<a>`), will have no attributes or whitespace. That means the body tag will always be "`<body>`", there will be no spaces or other characters. The same goes for the end tags.
  - 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=`".

- If you encounter a tag that is not valid then the page is not balanced.
- I will ensure all tags are lower case.
- To ignore whitespace, parse everything one character at a time ignoring space, tab, newline, and carriage return (`' '`, `'\t'`, `'\n'`, `'\r'`).

Hints

So far we have used arrays, linked lists, pointer, stacks, and queues. Chances are you will be using more that one data structure to solve this problem, but you will only need the ones listed.

Also, if you are familiar with the STL, you may use vector, stack, and queue from the STL library, but nothing else.

The C++ Regular Expression Library may be used for parsing but it isn't required.

Warning

The Data Structure projects are *significantly* more demanding than the labs. Start thinking about the data structure now! If you hit a roadblock (and you should/will), please come talk to me. When you talk to me be prepared to show me what you have thought of so far. If you come in with an idea, I will guide your idea to a good solution. If you come in without an idea, I will tell you to keep thinking about it. Keep in mind, there are many good, fair, and terrible solutions to Data Structure projects.

Write some test cases

Part of this project grade will be how well you can write test cases. You will be in charge of all test cases.

How to turn in

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

```
git add <files>
git commit
git push
```

Assigned

October 3, 2022

Due Date

October 31, 2022 11:59 PM

Teamwork

No teamwork, your work must be your own.