DSA (SE) Assignment 1– Fall 22

Deadline: 27 October, 2022

CLO 3 Assessment [10 Marks]

Background:
An important of application of stacks in which matching is important is in the validation of HTML documents. HTML is the standard format for hyperlinked documents on the Internet. In an HTML document, portions of text are delimited by ***HTML tags***. A simple opening HTML tag has the form "<name>" and the corresponding closing tag has the form "</name>." Commonly used HTML tags include:

- body: document body
- h1: section header
- center: center justify
- p: paragraph
- ol: numbered (ordered) list
- li: list item

Deliverable:
Your task is to write a program to check that the tags properly match for the supplied HTML document. The program should ask for the name of the HTML file and then displays the output in the form of opening and closing tags on standard output stream. In case of successful matching, program should display the matching sequence of tags on standard output stream. Otherwise, program should display the missing tags in the HTML document.

Penalty for Plagiarism: F grade will be assigned in the course as per university regulations.

Guidelines:
A very similar approach to that given in Code Fragment (given below) can be used to match the tags in an HTML document. We push each opening tag on a stack, and when we encounter a closing tag, we pop the stack and verify that the two tags match.

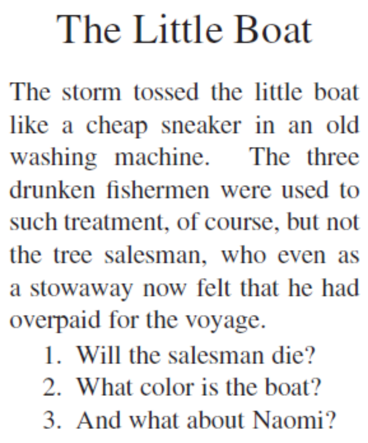
```
Algorithm ParenMatch($X,n$):
  Input: An array $X$ of $n$ tokens, each of which is either a grouping symbol, a
    variable, an arithmetic operator, or a number
  Output: true if and only if all the grouping symbols in $X$ match
  Let $S$ be an empty stack
  for $i \leftarrow 0$ to $n-1$ do
    if $X[i]$ is an opening grouping symbol then
      $S$.push($X[i]$)
    else if $X[i]$ is a closing grouping symbol then
      if $S$.empty() then
        return false          {nothing to match with}
      if $S$.top() does not match the type of $X[i]$ then
        return false          {wrong type}
      $S$.pop()
  if $S$.empty() then
    return true          {every symbol matched}
  else
    return false          {some symbols were never matched}
```

Code Fragment: Algorithm for matching grouping symbols in an arithmetic expression

A sample HTML document and a possible rendering in the figure below:

```
<body>
<center>
<h1> The Little Boat </h1>
</center>
<p> The storm tossed the little
boat like a cheap sneaker in an
old washing machine. The three
drunken fishermen were used to
such treatment, of course, but
not the tree salesman, who even
as a stowaway now felt that he
had overpaid for the voyage. </p>
<ol>
<li> Will the salesman die? </li>
<li> What color is the boat? </li>
<li> And what about Naomi? </li>
</ol>
</body>
```

(a)



The Little Boat

The storm tossed the little boat like a cheap sneaker in an old washing machine. The three drunken fishermen were used to such treatment, of course, but not the tree salesman, who even as a stowaway now felt that he had overpaid for the voyage.

1. Will the salesman die?
2. What color is the boat?
3. And what about Naomi?

(b)

Figure: HTML tags: (a) an HTML document; (b) its rendering.

Relevant code fragments for this task are given below. Further help is available in section 5.1.7 of Michael Goodrich book.

```
vector<string> getHtmlTags() {                   // store tags in a vector
  vector<string> tags;                           // vector of html tags
  while (cin) {                                  // read until end of file
    string line;
    getline(cin, line);                          // input a full line of text
    int pos = 0;                                 // current scan position
    int ts = line.find("<", pos);                // possible tag start
    while (ts != string::npos) {                 // repeat until end of string
      int te = line.find(">", ts+1);             // scan for tag end
      tags.push_back(line.substr(ts, te−ts+1)); // append tag to the vector
      pos = te + 1;                              // advance our position
      ts = line.find("<", pos);
    }
  }
  return tags;                                   // return vector of tags
}
```

Code Fragment: Get a vector of HTML tags from the input, and store them in a vector of strings.

```
                                                  // check for matching tags
bool isHtmlMatched(const vector<string>& tags) {
  LinkedStack S;                                  // stack for opening tags
  typedef vector<string>::const_iterator Iter;// iterator type
                                                  // iterate through vector
  for (Iter p = tags.begin(); p != tags.end(); ++p) {
    if (p−>at(1) != '/')                          // opening tag?
      S.push(*p);                                 // push it on the stack
    else {                                        // else must be closing tag
      if (S.empty()) return false;                // nothing to match - failure
      string open = S.top().substr(1);            // opening tag excluding '<'
      string close = p−>substr(2);                // closing tag excluding '</'
      if (open.compare(close) != 0) return false; // fail to match
      else S.pop();                               // pop matched element
    }
  }
  if (S.empty()) return true;                     // everything matched - good
  else return false;                              // some unmatched - bad
}
```

Code Fragment: Check whether HTML tags stored in the vector tags are matched.