6.1.5 Matching Parentheses and HTML Tags

In this subsection, we explore two related applications of stacks, both of which involve testing for pairs of matching delimiters. In our first application, we consider arithmetic expressions that may contain various pairs of grouping symbols, such as

- Parentheses: "(" and ")"
- Braces: "{" and "}"
- Brackets: "[" and "]"

Each opening symbol must match its corresponding closing symbol. For example, a left bracket, "[," must match a corresponding right bracket, "]," as in the following expression

$$[(5+x)-(y+z)].$$

The following examples further illustrate this concept:

- Correct: $()(()){([()])}$
- Correct: $((()(()){([()])}))$
- Incorrect: $)(()){([()])}$
- Incorrect: $(\{[])\}$
- Incorrect: $($

We leave the precise definition of a matching group of symbols to Exercise R-6.6.

An Algorithm for Matching Delimiters

An important task when processing arithmetic expressions is to make sure their delimiting symbols match up correctly. We can use a stack to perform this task with a single left-to-right scan of the original string.

Each time we encounter an opening symbol, we push that symbol onto the stack, and each time we encounter a closing symbol, we pop a symbol from the stack (assuming it is not empty) and check that these two symbols form a valid pair. If we reach the end of the expression and the stack is empty, then the original expression was properly matched. Otherwise, there must be an opening delimiter on the stack without a matching symbol. If the length of the original expression is n, the algorithm will make at most n calls to push and n calls to pop. Code Fragment 6.7 presents a Java implementation of such an algorithm. It specifically checks for delimiter pairs ( ), { }, and [ ], but could easily be changed to accommodate further symbols. Specifically, we define two fixed strings, `"({["` and `")}]"`, that are intentionally coordinated to reflect the symbol pairs. When examining a character of the expression string, we call the indexOf method of the String class on these special strings to determine if the character matches a delimiter and, if so, which one. Method indexOf returns the the index at which a given character is first found in a string (or -1 if the character is not found).

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>`". For example, we see the `<body>` tag on the first line of Figure 6.3a, and the matching `</body>` tag at the close of that document. Other commonly used HTML tags that are used in this example include:

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

Ideally, an HTML document should have matching tags, although most browsers tolerate a certain number of mismatching tags. In Code Fragment 6.8, we give a Java method that matches tags in a string representing an HTML document.

We make a left-to-right pass through the raw string, using index j to track our progress. The indexOf method of the String class, which optionally accepts a starting index as a second parameter, locates the `'<'` and `'>'` characters that define the tags. Method substring, also of the String class, returns the substring starting at a given index and optionally ending right before another given index. Opening tags are pushed onto the stack, and matched against closing tags as they are popped from the stack, just as we did when matching delimiters in Code Fragment 6.7.

```java
/** Tests if every opening tag has a matching closing tag in HTML string. */
public static boolean isHTMLMatched(String html) {
  Stack<String> buffer = new LinkedStack<>();
  int j = html.indexOf('<');                       // find first '<' character (if any)
  while (j != −1) {
    int k = html.indexOf('>', j+1);                // find next '>' character
    if (k == −1)
      return false;                                // invalid tag
    String tag = html.substring(j+1, k);           // strip away < >
    if (!tag.startsWith("/"))                      // this is an opening tag
      buffer.push(tag);
    else {                                         // this is a closing tag
      if (buffer.isEmpty())
        return false;                              // no tag to match
      if (!tag.substring(1).equals(buffer.pop()))
        return false;                              // mismatched tag
    }
    j = html.indexOf('<', k+1);                    // find next '<' character (if any)
  }
  return buffer.isEmpty();                         // were all opening tags matched?
}
```

Code Fragment 6.8: Method for testing if an HTML document has matching tags.