591. Tag Validator

Description Hints Submissions Solutions

- Total Accepted: **421**
- Total Submissions: **2176**
- Difficulty: **Hard**
- Contributors:fallcreek

Given a string representing a code snippet, you need to implement a tag validator to parse the code and return whether it is valid. A code snippet is valid if all the following rules hold:

1. The code must be wrapped in a **valid closed tag**. Otherwise, the code is invalid.
2. A **closed tag** (not necessarily valid) has exactly the following format : `<TAG_NAME>TAG_CONTENT</TAG_NAME>`. Among them, `<TAG_NAME>` is the start tag, and `</TAG_NAME>` is the end tag. The TAG_NAME in start and end tags should be the same. A closed tag is **valid** if and only if the TAG_NAME and TAG_CONTENT are valid.
3. A **valid** `TAG_NAME` only contain **upper-case letters**, and has length in range [1,9]. Otherwise, the `TAG_NAME` is **invalid**.
4. A **valid** `TAG_CONTENT` may contain other **valid closed tags**, **cdata** and any characters (see note1) **EXCEPT** unmatched `<`, unmatched start and end tag, and unmatched or closed tags with invalid TAG_NAME. Otherwise, the `TAG_CONTENT` is **invalid**.
5. A start tag is unmatched if no end tag exists with the same TAG_NAME, and vice versa. However, you also need to consider the issue of unbalanced when tags are nested.
6. A `<` is unmatched if you cannot find a subsequent `>`. And when you find a `<` or `</`, all the subsequent characters until the next `>` should be parsed as TAG_NAME (not necessarily valid).
7. The cdata has the following format : `<![CDATA[CDATA_CONTENT]]>`. The range of `CDATA_CONTENT` is defined as the characters between `<![CDATA[` and the **first subsequent** `]]>`.

8. CDATA_CONTENT may contain **any characters**. The function of cdata is to forbid the validator to parse CDATA_CONTENT, so even it has some characters that can be parsed as tag (no matter valid or invalid), you should treat it as **regular characters**.

Valid Code Examples:

Input: "<DIV>This is the first line <![CDATA[<div>]]></DIV>"
Output: True
Explanation:

The code is wrapped in a closed tag : <DIV> and </DIV>.

The TAG_NAME is valid, the TAG_CONTENT consists of some characters and cdata.

Although CDATA_CONTENT has unmatched start tag with invalid TAG_NAME, it should be co nsidered as plain text, not parsed as tag.

So TAG_CONTENT is valid, and then the code is valid. Thus return true.

Input: "<DIV>>> ![cdata[]] <![CDATA[<div>]>]]>]]>>]</DIV>"
Output: True
Explanation:

We first separate the code into : start_tag|tag_content|end_tag.

start_tag -> **"<DIV>"**

end_tag -> **"</DIV>"**

tag_content could also be separated into : text1|cdata|text2.

text1 -> **">> ![cdata[]] "**

cdata -> **"<![CDATA[<div>]>]]>"**, where the CDATA_CONTENT is **"<div>]>"**

text2 -> **"]]>>]"**

The reason why start_tag is NOT **"<DIV>>>"** is because of the rule 6.

The reason why cdata is NOT **"<![CDATA[<div>]>]]>]]>"** is because of the rule 7.

Invalid Code Examples:

Input: "<A> <B> </A> </B>"**Output:** False**Explanation:** Unbalanced. If "<A>" is closed, then "<B>" must be unmatched, and vice versa.

Input: "<DIV> div tag is not closed <DIV>"**Output:** False

Input: "<DIV> unmatched < </DIV>"**Output:** False

Input: "<DIV> closed tags with invalid tag name 123 </DIV>"**Output:** False

Input: "<DIV> unmatched tags with invalid tag name </1234567890> and <CDATA[[]]> </DIV>"**Output:** False

Input: "<DIV> unmatched start tag <B> and unmatched end tag </C> </DIV>"**Output:** False

Note:

1. For simplicity, you could assume the input code (including the **any characters** mentioned above) only contain letters, digits, '<','>','/','!','[',']' and ' '.

Subscribe to see which companies asked this question.

Show Tags

```
#include<iostream>
#include<sstream>
#include<stdio.h>
#include<vector>
#include<unordered_set>
#include<unordered_map>
#include<limits.h>
#include<set>
#include<random>
```

```
#include<ctime>
#include<stack>
#include<string>
using namespace std;

bool startwith(string code, string with)
{
    int ans = code.find(with);
    return ans==0;
}

bool endwith(string code, string with)
{
    int ans = code.find(with);
    return ans == (int)code.size()-(int)with.size();
}

bool validTag(string tag)
{
    if(!startwith(tag,"<") || !endwith(tag,">")) return false;
    if (tag.size()<3 || tag.size()>11) return false;
    for (char kk:tag.substr(1,tag.size()-2))
    {
        if(!isupper(kk)) return false;
    }
    //cout <<"I am here!"<<endl;
    return true;
}

bool isValid(string code) {
    if(!startwith(code,"<")) return false;
    if(code.size()<2 || !isupper(code[1])) return false;
    stack<string> st;
    string cur = ""; int i=0;
    while(i<(int)code.size())
    {
        cur += code[i];
        if(startwith(cur,"<![CDATA["))
        {
            if(endwith(cur,"]]>"))
            {
                cur = "";
            }
        }else if(startwith(cur,"</"))
```

```
        {
            if(endwith(cur,">"))
            {
                cout << cur << endl;
                string tag = cur.substr(2,cur.size()-3);
                if(st.empty()||st.top()!="<"+tag+">") return false;
                st.pop();
                cur = "";
//Be careful of
//"<A></A><B></B>" case. The entire code has to be exact ONE valid closed
tag.
                if(i<(int)code.size()-1 && st.empty()) return false;
            }
        }else if(startwith(cur,"<"))
        {
            if(endwith(cur,">"))
            {
                if(validTag(cur)==false) return false;
                st.push(cur);
                cur = "";
            }

        }else{
            cur = "";
        }
        i++;
    }
    cout << "I am here"<<endl;
    return cur=="" && st.empty();
}

int main(int argc,char *argv[])
{
    string test = "<DIV>This is the first line <![CDATA[<div>]]></DIV>";
    string test2= "<A></A><B></B>";
    bool ans = isValid(test2);
    cout << ans << endl;
    return 0;
}
```