Valid Number

Validate if a given string is numeric.

Some examples:
`"0"` => `true`
`" 0.1 "` => `true`
`"abc"` => `false`
`"1 a"` => `false`
`"2e10"` => `true`

Note: It is intended for the problem statement to be ambiguous. You should gather all requirements up front before implementing one.

Update (2015-02-10):
The signature of the `C++` function had been updated. If you still see your function signature accepts a `const char *` argument, please click the reload button🔄 to reset your code definition.

Solution 1

The description do not give a clear explantion of the definition of a valid Number, we just use more and more trick to get the right solution. It's too bad, it's waste of my time

written by aqin original link here

Solution 2

The idea is pretty straightforward. A valid number is composed of the significand and the exponent (which is optional). As we go through the string, do the following things one by one:

1. skip the leading whitespaces;
2. check if the significand is valid. To do so, simply skip the leading sign and count the number of digits and the number of points. A valid significand has no more than one point and at least one digit.
3. check if the exponent part is valid. We do this if the significand is followed by 'e'. Simply skip the leading sign and count the number of digits. A valid exponent contain at least one digit.
4. skip the trailing whitespaces. We must reach the ending 0 if the string is a valid number.

==========

```
bool isNumber(const char *s)
{
    int i = 0;

    // skip the whilespaces
    for(; s[i] == ' '; i++) {}

    // check the significand
    if(s[i] == '+' || s[i] == '-') i++; // skip the sign if exist

    int n_nm, n_pt;
    for(n_nm=0, n_pt=0; (s[i]<='9' && s[i]>='0') || s[i]=='.'; i++)
        s[i] == '.' ? n_pt++:n_nm++;
    if(n_pt>1 || n_nm<1) // no more than one point, at least one digit
        return false;

    // check the exponent if exist
    if(s[i] == 'e') {
        i++;
        if(s[i] == '+' || s[i] == '-') i++; // skip the sign

        int n_nm = 0;
        for(; s[i]>='0' && s[i]<='9'; i++, n_nm++) {}
        if(n_nm<1)
            return false;
    }

    // skip the trailing whitespaces
    for(; s[i] == ' '; i++) {}

    return s[i]==0;  // must reach the ending 0 of the string
}
```

written by GuaGua original link here

Solution 3

All we need is to have a couple of flags so we can process the string in linear time:

```
public boolean isNumber(String s) {
    s = s.trim();

    boolean pointSeen = false;
    boolean eSeen = false;
    boolean numberSeen = false;
    boolean numberAfterE = true;
    for(int i=0; i<s.length(); i++) {
        if('0' <= s.charAt(i) && s.charAt(i) <= '9') {
            numberSeen = true;
            numberAfterE = true;
        } else if(s.charAt(i) == '.') {
            if(eSeen || pointSeen) {
                return false;
            }
            pointSeen = true;
        } else if(s.charAt(i) == 'e') {
            if(eSeen || !numberSeen) {
                return false;
            }
            numberAfterE = false;
            eSeen = true;
        } else if(s.charAt(i) == '-' || s.charAt(i) == '+') {
            if(i != 0 && s.charAt(i-1) != 'e') {
                return false;
            }
        } else {
            return false;
        }
    }

    return numberSeen && numberAfterE;
}
```

We start with trimming.

- If we see `[0-9]` we reset the number flags.
- We can only see `.` if we didn't see `e` or `.`.
- We can only see `e` if we didn't see `e` but we did see a number. We reset numberAfterE flag.
- We can only see `+` and `-` in the beginning and after an `e`
- any other character break the validation.

At the and it is only valid if there was at least 1 number and if we did see ar `e` then a number after it as well.

So basically the number should match this regular expression:

`[-+]?[0-9]*(.[0-9]+)?(e[-+]?[0-9]+)?`

written by balint original link here

From Leetcoder.