Valid Parentheses

Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

Open brackets must be closed by the same type of brackets.
Open brackets must be closed in the correct order.
Note that an empty string is also considered valid.

Example 1:

Input: "()"
Output: true
Example 2:

Input: "()[]{}"
Output: true
Example 3:

Input: "(]"
Output: false
Example 4:

Input: "([)]"
Output: false
Example 5:

Input: "{[]}"
Output: true

Solution in C++

```
class Solution {
public:
    int iAction(char ch) {
        switch(ch) {
            case '(': return 1;
            case '{': return 2;
            case '[': return 3;
            case ')': return -1;
            case '}': return -2;
            case ']': return -3;
        }
        return 0; // no action needed
    }
    bool isValid(string s) {
        stack<char> myStack;

        for(int i=0; i<s.length(); i++) {
            int a=iAction(s[i]);
            if (a>0) myStack.push(s[i]);
            else {
                if (myStack.empty()) return false;

                int previousCase=iAction(myStack.top());
                if ((int)fabs(a)!=previousCase) return false;
                myStack.pop();
            }
        }
        return (myStack.empty());

    }
};
```

Solution in Java

```
class Solution {

  // Hash table that takes care of the mappings.
  private HashMap<Character, Character> mappings;

  // Initialize hash map with mappings. This simply makes the code easier to read.
  public Solution() {
    this.mappings = new HashMap<Character, Character>();
    this.mappings.put(')', '(');
    this.mappings.put('}', '{');
    this.mappings.put(']', '[');
  }

  public boolean isValid(String s) {

    // Initialize a stack to be used in the algorithm.
    Stack<Character> stack = new Stack<Character>();

    for (int i = 0; i < s.length(); i++) {
      char c = s.charAt(i);

      // If the current character is a closing bracket.
      if (this.mappings.containsKey(c)) {

        // Get the top element of the stack. If the stack is empty, set a dummy value of '#'
        char topElement = stack.empty() ? '#' : stack.pop();

        // If the mapping for this bracket doesn't match the stack's top element, return false.
        if (topElement != this.mappings.get(c)) {
          return false;
        }
      } else {
        // If it was an opening bracket, push to the stack.
        stack.push(c);
      }
    }

    // If the stack still contains elements, then it is an invalid expression.
    return stack.isEmpty();
  }
}
```

Solution in Python

```
class Solution(object):
    def isValid(self, s):
        """
        :type s: str
        :rtype: bool
        """

        # The stack to keep track of opening brackets.
        stack = []

        # Hash map for keeping track of mappings. This keeps the code very clean.
        # Also makes adding more types of parenthesis easier
        mapping = {")": "(", "}": "{", "]": "["}

        # For every bracket in the expression.
        for char in s:

            # If the character is an closing bracket
            if char in mapping:

                # Pop the topmost element from the stack, if it is non empty
                # Otherwise assign a dummy value of '#' to the top_element variable
                top_element = stack.pop() if stack else '#'

                # The mapping for the opening bracket in our hash and the top
                # element of the stack don't match, return False
                if mapping[char] != top_element:
                    return False
            else:
                # We have an opening bracket, simply push it onto the stack.
                stack.append(char)

        # In the end, if the stack is empty, then we have a valid expression.
        # The stack won't be empty for cases like ((((
        return not stack
```