1. Java Stack

Given a list of strings of bracket characters: *{}()*, the string of brackets is *balanced* under the following conditions:

1. It is the empty string.
2. If strings *a* and *b* are balanced, then *ab* is balanced.
3. If string *a* is balanced, then *(a)* and *{a}* are balanced.

Write a class that determines whether the brackets in each string are balanced and returns *true* if the string is balanced, or *false* if it is not.

Example 0

s = [*"{}()", "{()}", "(({()))"]*

s[0] exhibits condition *2* above. "{}" and "()" are balanced, so "{}()" is balanced. Return *true*.
s[1] exhibits condition 3 above. "()" is balanced, so "{()}" is balanced. Return *true*.
s[2] exhibits condition 3 above. "()" is balanced, so "{()}" is balanced and "(({()))" is balanced. Return *true*.

Example 1

s = [*"{}(", "{()}", "((", "}{"]*

s[0] rarr *2*. *"{}("* is an unbalanced string due to the open *"("*. Return *false*.
s[1] rarr 2. *"{()}"* is an unbalanced string due to ")" before "{" has been closed. Return *false*.
s[2] rarr 2. *"(("*, is an unbalanced string because neither "(" is closed. Return *false*.
s[2] rarr 2. *"}{"* is an unbalanced string because "}" comes before a "{" and because the final "{" is not closed. Return *false*.

Function Description

The provided code contains the declaration for a class named *Solution* with a *main* method that does the following:

- Creates a *Parser* object.
- Reads an unknown number of strings from stdin.
- Passes each string as an argument to the *Parser* object's *isBalanced* method and

Function Description

The provided code contains the declaration for a class named *Solution* with a *main* method that does the following:

- Creates a *Parser* object.
- Reads an unknown number of strings from stdin.
- Passes each string as an argument to the *Parser* object's *isBalanced* method and prints value returned by the method on a new line.

Complete the function an *isBalanced* in the editor below.

isBalanced has the following parameter(s):

string s: a string of characters to check for balance

Returns :

bool : a boolean that denotes whether the string is balanced: *true* if the string is balanced, or *false* if it is not

Constraints

- Each string consists only of the characters *{*, *}*, *(*, and *)*.
- Each string has fewer than 50 characters.

▼ Input Format for Custom Testing

Input from stdin will be processed as follows and passed to your Parser.isBalanced method.

Each line contains a string to parse.

▼ Sample Case 0

Sample Input 0

```
STDIN       Function
-----       --------
{}()    →   s = '{}()', '({()})', '{(('
({()})
{((
```

Sample Output 0

Input Format for Custom Testing

Input from stdin will be processed as follows and passed to your Parser.isBalanced method.

Each line contains a string to parse.

Sample Case 0

Sample Input 0

```
STDIN       Function
-----       --------
{}()    →   s = ['{}())', '({()})', '{}(']
({()})
{}(
```

Sample Output 0

```
true
true
false
```

Explanation 0

2. *'{}()'* contains two adjacent balanced strings, *'{}'* and *'()'*, so return *true*.

3. *'({()})'* contains a balanced string, *'()'*, nested inside another balanced string, *'{}'*, nested inside another balanced string, *'()'*. Return *true*.

2. *'{}('* contains a balanced string *'{}'*, followed by an unbalanced string *'('*. Return *false*.

13m left

ALL

2. How Will You Compare?

Write a *Comparator* class with the following *3* overloaded *compare* methods:

1. *boolean compare(int a, int b)*: Return *true* if *int a = int b*, otherwise return *false*.
2. *boolean compare(string a, string b)*: Return *true* if *string a = string b*, otherwise return *false*.
3. *boolean compare(int[] a, int[] b)*: Return *true* if both of the following conditions hold *true*:
   - Arrays *a* and *b* are of equal length.
   - For each index *i* (where $0 \le i < |a|, |b|$), $a[i] = b[i]$.

   Otherwise, return *false*.

Note: For *C++*, both parameters are of type *Vector<int>*.

Constraints

- For strings, $1 \le |a|, |b| \le 2000$
- For integers, $0 \le a, b \le 10000000$
- For integer arrays, $0 \le |a|, |b| \le 10$

▼ Input Format for Custom Testing

Input from stdin will be processed as follows and passed to the function.

The first line contains an integer *T*, the number of test cases.
Each of the next *T* sets of lines is in one of the following formats:

- The first line contains the integer ~~*t*~~ *representing the comparison type (1, 2 or 3 for int, string or array comparison respectively).*
  The next two lines contain strings *a* and *b*.
- The first line contains the integer ~~*t*~~ *representing the overloaded function type.*
  The next two lines contain integers *a* and *b*.
- The first line contains the integer ~~*t*~~ *representing the overloaded function type.*
  The next three lines contain the following:
  1. Two space-separated integers ~~*ta*~~ n and ~~*tb*~~ *m*, the lengths of arrays *a* and *b*.
  2. A line of ~~*ta*~~ n space-separated integers *a[i]*.
  3. A line of ~~*tb*~~ *m* space-separated integers *b[i]*.

▼ Sample Case 0

Sample Input 0

Test Result

Type here to search

- The first line contains the integer ~~3~~ *representing the overloaded function type.*
  The next three lines contain the following:
  1. Two space-separated integers ~~*la*~~ n and ~~*lb*~~ *m*, the lengths of arrays *a* and *b*.
  2. A line of ~~*la*~~ n space-separated integers *a[i]*.
  3. A line of ~~*lb*~~ *m* space-separated integers *b[i]*.

▼ Sample Case 0

Sample Input 0

```
STDIN           Function
-----           --------
3           →   T = 3 number of test cases.
1           →   Comparison type 1
hello world →   a = "hello world"
hello world →   b = "hello world"
2           →   Comparison type 2
3           →   a = 3
4           →   b = 4
3           →   Comparison type 3
3 3         →   a[] size n=3  b[] size m=3
1 2 3       →   a = [1, 2, 3]
1 2 3       →   b = [1, 2, 3]
```

Sample Output 0

```
Same
Different
Same
```

Explanation 0

There are 3 test cases:

Test Case	comparison type	a	b	Output	Explanation
1	1	"hello world"	"hello world"	"Same"	Both strings are the same.
				"Diff"	The two integers are different 1 ≠

Language

Test Result

ALL

3. What is the output of the following program:

What is the output of the following program:

```
interface BaseI { void method(); }

class BaseC
{
   public void method()
   {
      System.out.println("Inside BaseC::method");
   }
}

class ImplC extends BaseC implements BaseI
{
   public static void main(String []s)
   {
      (new ImplC()).method();
   }
}
```

Pick **ONE** option

- ○ null
- ○ Compilation fails
- ● Inside BaseC::method
- ○ None of the above

Clear Selection

Saved!

4. Which is the right answer to the following?

Given the follow code, which are true? (Choose all that apply):

```
1. interface Syrupable {
2. void getSugary();
3. }
4. abstract class Pancake implements Syrupable { }
5.
6. class BlueBerryPancake implements Pancake {
7.     public void getSugary() { ; }
8. }
9. class SourdoughBlueBerryPancake extends BlueBerryPancake {
10.     void getSugary(int s) { ; }
11. }
```

Pick **ONE** option

- ○ Compilation succeeds.
- ○ Compilation fails due to an error on line 2.
- ◉ Compilation fails due to an error on line 4.
- ○ Compilation fails due to an error on line 6.
- ○ Compilation fails due to an error on line 7.
- ○ Compilation fails due to an error on line 9.
- ○ Compilation fails due to an error on line 10.

Clear Selection

Saved!

ALL

5. Find the Output

What will the output be?

```
try
{
    Float f1 = new Float("3.0");
    int x = f1.intValue();
    byte b = f1.byteValue();
    double d = f1.doubleValue();
    System.out.println(x + b + d);
}
catch (NumberFormatException e) /* Line 9 */
{
    System.out.println("bad number"); /* Line 11 */
}
```

Pick **ONE** option

- (●) 9.0
- ( ) bad number
- ( ) Compilation fails on line 9
- ( ) Compilation fails on line 11

Clear Selection

6. Which of the following is not a correct way of commenting?

Which of the following is **not** a correct way of commenting?

Clear Selection

ALL

1

2

3

4

5

6

6. Which of the following is not a correct way of commenting?

Which of the following is **not** a correct way of commenting?

Pick **ONE** option

- (●) /* Here is a comment **** */
- ( ) /* This is also a comment /* More comments */ */
- ( ) /* This is also a comment // More comments */
- ( ) // /* This is a // // comment */

Clear Selection

Continue