Exception Safe Stack & Queue

Part 1.

1. Implement an exception safe queue like we did for stack.
2. Implement a queue but this time it should be based on a container as a template parameter. The signature should be like this:
   `template<class T, class Container = std::vector<T>>`
   `class _queue;`
   You don't need to do manual new and delete, you should call the appropriate functions (`push_back, push_front, pop_back, pop_front`, etc.) of the underlying container. The underlying container can be any of STD's containers or any of the containers we implemented during previous classes

Part 2.

All the 4 tests: _test1, _test2, _test3 and _test4 are in the "tests2.h" header file.

Problem 1. Valid Parentheses

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

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.

Example 1:

```
Input: s = "()"
Output: true
```

Example 2:

```
Input: s = "()[]{}"
Output: true
```

Example 3:

```
Input: s = "(]"
Output: false
```

Problem 2. BS Tree Iterator (using stack)

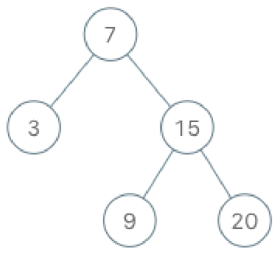
Implement the BSTIterator class that represents an iterator over the in-order traversal of a binary search tree (BST):

- BSTIterator(TreeNode root) Initializes an object of the BSTIterator class. The root of the BST is given as part of the constructor. The pointer should be initialized to a non-existent number smaller than any element in the BST.
- boolean hasNext() Returns true if there exists a number in the traversal to the right of the pointer, otherwise returns false.
- int next() Moves the pointer to the right, then returns the number at the pointer.

Notice that by initializing the pointer to a non-existent smallest number, the first call to next() will return the smallest element in the BST.

You may assume that next() calls will always be valid. That is, there will be at least a next number in the in-order traversal when next() is called.

Example 1:



```
Input
["BSTIterator", "next", "next", "hasNext", "next", "hasNext", "next", "hasNext", "next", "hasNext"]
[[[7, 3, 15, null, null, 9, 20]], [], [], [], [], [], [], [], [], []]
Output
[null, 3, 7, true, 9, true, 15, true, 20, false]

Explanation
BSTIterator bSTIterator = new BSTIterator([7, 3, 15, null, null, 9, 20]);
bSTIterator.next();    // return 3
bSTIterator.next();    // return 7
bSTIterator.hasNext(); // return True
bSTIterator.next();    // return 9
bSTIterator.hasNext(); // return True
bSTIterator.next();    // return 15
bSTIterator.hasNext(); // return True
bSTIterator.next();    // return 20
bSTIterator.hasNext(); // return False
```

Problem 3. Basic Calculator

Given a string s which represents an expression, evaluate this expression and return its value.
The integer division should truncate toward zero.
You may assume that the given expression is always valid. All intermediate results will be in the range of [-231, 231 - 1].

Example 1:

```
Input: s = "3+2*2"
Output: 7
```

Example 2:

```
Input: s = " 3/2 "
Output: 1
```

Example 3:

```
Input: s = " 3+5 / 2 "
Output: 5
```

Problem 4. Winning Game

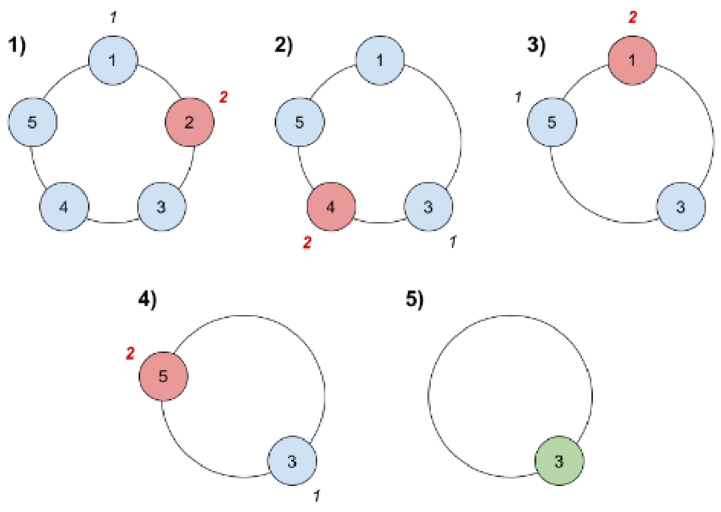
There are n friends that are playing a game. The friends are sitting in a circle and are numbered from 1 to n in clockwise order. More formally, moving clockwise from the ith friend brings you to the (i+1)th friend for 1 <= i < n, and moving clockwise from the nth friend brings you to the 1st friend.

The rules of the game are as follows:

1. Start at the 1st friend.
2. Count the next k friends in the clockwise direction including the friend you started at. The counting wraps around the circle and may count some friends more than once.
3. The last friend you counted leaves the circle and loses the game.
4. If there is still more than one friend in the circle, go back to step 2 starting from the friend immediately clockwise of the friend who just lost and repeat.
5. Else, the last friend in the circle wins the game.

Given the number of friends, n, and an integer k, return *the winner of the game*.

Example 1:



```
Input: n = 5, k = 2
Output: 3
Explanation: Here are the steps of the game:
1) Start at friend 1.
2) Count 2 friends clockwise, which are friends 1 and 2.
3) Friend 2 leaves the circle. Next start is friend 3.
4) Count 2 friends clockwise, which are friends 3 and 4.
5) Friend 4 leaves the circle. Next start is friend 5.
6) Count 2 friends clockwise, which are friends 5 and 1.
7) Friend 1 leaves the circle. Next start is friend 3.
8) Count 2 friends clockwise, which are friends 3 and 5.
9) Friend 5 leaves the circle. Only friend 3 is left, so they are the winner.
```

Example 2:

```
Input: n = 6, k = 5
Output: 1
Explanation: The friends leave in this order: 5, 4, 6, 2, 3. The winner is friend 1.
```

Could you solve this problem in linear time with constant space?