

Sergio E. Garcia Tapia

Algorithms by Sedgewick and Wayne (4th edition) [SW11]

September 22nd, 2024

1.3: Bags, Queues, and Stacks

Exercise 1. Add a method `isFull()` to `FixedCapacityStackOfStrings`.

Solution. See the `com.segarciat.algs4.ch1.sec3.ex1.FixedCapacityStackOfStrings` class.

Exercise 2. Give the output printed by `java Stack` for the input

```
it was - the best - of times - - - it was - the - -
```

Solution. The `-` causes the latest added word to be removed. The contents of the stack at each step are as follows:

```
it
it was
it
it the
it the best
it the
it the of
it the of times
it the of
it the
it
it it
it it was
it it
it it the
it it
it
```

The output is the last line, `it`.

Exercise 3. Suppose that a client performs an intermixed sequence of (stack) *push* and *pop* operations. The push operations put the integers 0 through 9 in order onto the stack; the pop operations print out the return values. Which of the following sequence(s) could *not* occur?

- (a) 4 3 2 1 0 9 8 7 6 5
- (b) 4 6 8 7 5 3 2 9 0 1
- (c) 2 5 6 7 4 8 9 3 1 0
- (d) 4 3 2 1 0 5 6 7 8 9

- (e) 1 2 3 4 5 6 9 8 7 0
- (f) 0 4 6 5 3 8 1 7 2 9
- (g) 1 4 7 9 8 6 5 3 0 2
- (h) 2 1 4 3 6 5 8 7 9 0

Solution.

- (a) Valid. This sequence involves pushing 0 through 4, then popping five times. Then, pushing 5 through 9, and then popping five times.
- (b) Invalid. The sequence involves pushing 0 through 4 and pop once to print 4. Then, we push 5 and 6, and pop once to print 6. Next, push 7 and 8 and then pop 8, and pop 7. Popping again would give 5. Popping again yields 3, and then 2. We can then push 9 and pop it. At this point we've got 0 and 1 left on the stack. The next item popped should be 1, so this sequence must be incorrect.
- (c) Valid. We push 0, 1, 2, then pop 2. Next, we push 3, 4, 5, and pop 5. Next, we push 6 and pop it, then push 7 and pop it. We pop next (4). Next we push 8 and pop it, push 9 and pop it. Next we pop 3. Finally, we pop 1 and 0.
- (d) Valid. We push 0, 1, 2, 3, and 4, then pop them all off, so the stack is empty. Next, we push 5 and pop it, push 6 and pop it, push 7 and pop it, push 8 and pop it, and push 9 and pop it.
- (e) Valid. For inputs 0, 1, 2, 3, 4, 5, 6, we push and immediately pop. Then we push 7, 8, 9, and pop 4 times.
- (f) Invalid. We push 0 and pop. Then, we push 1, 2, 3, 4 and pop 4. We now push 5 and 6, then we pop 6, 5, and 3. We push 7 and 8 and pop 8. If we pop next, we should get 2 from the stack, which does not match the next value in the sequence (1).
- (g) Invalid. We push 0 and 1, then pop 1. We push 2, 3, 4, then pop 4. We push 5, 6, 7, then pop 7. We push 8 and 9. Now we pop 9, pop 8, pop 6, pop 6, pop 5, and the next pop operation would be 2, but the sequence says 0.
- (h) Valid. We push 0, 1, 2, and pop 2 and 1. We push 3 and 4, then pop both. We push 5 and 6, then pop both. We push 7 and 8, then pop both. We push 9 then pop it immediately. Number 0 remains, and we indeed pop it.

Exercise 4. Write a stack client `Parentheses` that reads in a text stream from standard input and uses a stack to determine whether its parentheses are properly balanced. For example, your program should print `true` for `[()]{ }{ [()] () }` and `false` for `[(])`.

Solution. See the `com.segarciat.algs4.ch1.sec3.ex04.Parentheses` class.

Exercise 5. What does the following code fragment print when `n` is 50? Give a high-level description of what it does when presented with a positive integer `n`.

```
Stack<Integer> Stack = new Stack<Integer>();
while (n > 0)
{
    stack.push(n % 2);
    n = n / 2;
}
for (int d: stack) StdOut.print(d);
StdOut.println();
```

Solution. It prints the binary representation of n .

Exercise 6. What does the following code fragment do to the queue q ?

```
Stack<String> stack = new Stack<String>();
while (!q.isEmpty())
    stack.push(q.dequeue());
while (!stack.isEmpty())
    q.enqueue(stack.pop());
```

Solution. The fragment reverses the order of the entries in the queue q .

Exercise 7. Add a method `peek()` to `Stack` that returns the most recently inserted item on the stack (without popping it).

Solution. See the `com.segarciat.algs4.ch1.sec3.ex07.Stack` class.

References

- [SW11] Robert Sedgewick and Kevin Wayne. *Algorithms*. 4th ed. Addison-Wesley, 2011.
ISBN: 9780321573513.