The Stack
(Programming Club 3)

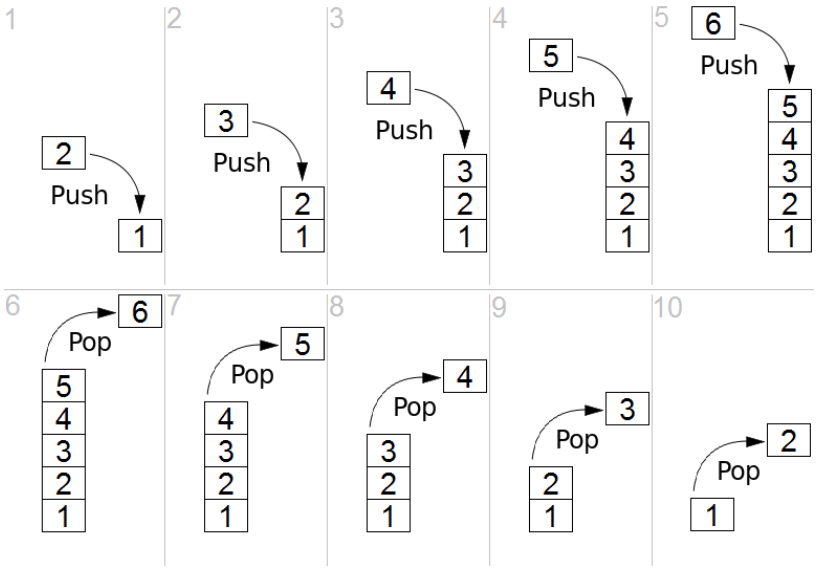
Mihai Enache – M.I.Enache@sms.ed.ac.uk, UG4, School of Informatics
15 October 2018

1. Introduction

Similar to an array, a stack is a collection of elements. However, it allows different types of operations that can be performed on it. You have probably encountered already the idea of a stack in the real life. Think about the case in which you have a heap of plates placed on top of each other as in the picture below:

If you want to reach a specific plate, say the 3rd one from top to bottom, you must first take out the first 2 plates from the pile and only then you can take your target plate. Also, if you want to add a new plate in the heap, you can only put it on the top of the others. Basically, in computer science, a stack is an abstract data type that emulates the same behaviour. Initially the stack is empty and you can perform 2 type of operations:

1) push x – add element x on top of the stack
2) pop – take out the element on top of the stack (must ensure the stack is not empty!)



(source: Wikipedia)

2. Applications

One of the basic applications of a stack is to determine whether a sequence of parenthesis (round brackets) is valid. A sequence of parenthesis is valid if it can be formed following the rules:

1) The empty string (“”) is a valid sequence of parenthesis
2) If A and B are valid sequences, then AB is also valid
3) If A is a valid sequence, then (A) is also valid

For example “()()”, “(())()” and “((()(()))” are valid, while “())” and “()((” are not. So how we can use the idea of a stack to determine whether a sequence is valid or not? We observe that a sequence is valid if for every ‘)’ there is a matching ‘(‘ on its left that hasn’t been assigned to another ‘)’. It also must be the case that there is a direct correspondence between every ‘)’ and its ‘(‘.

Thus, we can think of a solution like this: we traverse the sequence from left to right and whenever we encounter a ‘(‘, we just store it and wait to find the first ‘)’ that can be assigned to it. When we find a ‘)’, we are interested in taking the last ‘(‘ that was not assigned and pair it up with the current bracket, then remove it from the list of ‘(‘. The sequence can become invalid in 2 ways: we can either encounter a ‘)’ that has no corresponding ‘(‘, or we can finish the sequence and there are still some ‘(‘ that haven’t been matched. We notice that we are actually not interested in the position of the open brackets, but only in how many we found so far. Thus we can only simulate the size of the stack and still be able to solve the problem. The solution in pseudo-code is given below:

```
boolean is_valid_sequence(seq):
    num_open_brackets = 0, n = length(seq)
    for i = 0 to n – 1:
        if seq[i] == ‘(‘:
            num_open_brackets += 1
        else if seq[i] == ‘)’:
            if num_open_brackets > 0:
                num_open_brackets -= 1
            else:
                print “The sequence is invalid. ‘)’ at index “ + i + “ has no matching pair.”
                return false

    if num_open_brackets > 0:
        print “The sequence is invalid. Some ‘(‘ are unmatched.”
        return false

    return true
```

A slightly more difficult problem is to allow different type of brackets. Think about how you can extend the above solution to determine whether a given sequence containing the characters ‘(‘, ‘)’, ‘[‘, ‘]’, ‘{‘, ‘}’ is valid or not. This is also the first problem in the HackerRank Interview Preparation Kit set for stacks and queues (we will discuss about the queue next week, but for now you can try to solve the stack problems).

3. Resources

1. https://en.wikipedia.org/wiki/Stack_(abstract_data_type)#Applications_of_stacks
2. https://www.hackerrank.com/interview/interview-preparation-kit/stacks-queues/challenges
3. Extension: if you are familiar with object-oriented programming, implement a Stack class using one of the previous 2 data structures discussed (array and linked list).