Data Structure

Lab 1. Stack

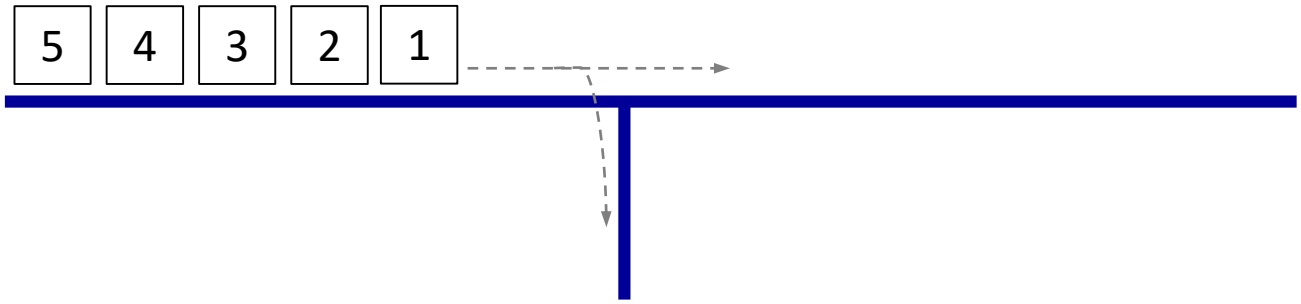
Shin Hong

Apr 3, 2020

Submission

- Deadline: 4 PM, Apr 7 (Tue)
- Online test
  - http://34.84.71.80
  - Your username and password can be found at Hisnet
  - Your file upload is counted as submission
- Report (optional)
  - Submit it to the homework repository at Hisnet (by 4 PM, Apr 7)
- Note
  - You must use stack.c used in the class
    https://github.com/hongshin/DataStructure/blob/stack/ver2/stack.c
  - You must construct a solution as a single source file

Railroad (1/2)



As shown in the figure above, railroad cars numbered 1, 2, ... , *N* are coming in order from the left. There is a train station with a single track where at most *M* numbers of trains can stay at a time.

A car may or may not enter the station at a moment if the number of cars in the station is less than *C*. A car cannot enter the station when *C* cars are already in the station. If a car had entered, it can exit the station only when no car that entered the station later remains in the station. Note that cars may reach to the right end in various orderings depending on whether a car enters the station or not. Also, note that some orderings are possible yet some orderings are impossible.

Write a program that determines whether it is possible to make cars reach to the right end in a certain ordering, or not.

Railroad (2/2)

- Input
  - Given from the standard input
  - The first line has two integers *N* and *C* for $0 \leq N \leq 1000$ and $1 \leq C \leq N$
  - The second line has *N* integers (*1* to *N*) whose sequence represents the expected ordering of cars
- Output
  - Print "`true`" if the expected ordering is possible
  - Print "`false`" if the expected ordering is impossible
- Example

```
5 3
2 4 3 5 1
```

<Input 1>

```
true
```

<Output 1>

```
7 2
3 5 4 2 6 7 1
```

<Input 2>

```
false
```

<Output 2>

Bracket (1/2)

We call that brackets *well balanced* when every opening bracket has a corresponding closing one, and the pairs of brackets are properly nested.

For well balanced brackets, we can define its *maximum depth* as the most number of brackets that surround a pair of bracket plus 1. For example, the maximum depth of (()(())) is 3 because the orange has the most surrounding brackests which are the green and the blue (3=2+1).

Write a program that determine the maximum depth of given brackets that consist of (,), {, }, <, >, [, and].

Bracket (2/2)

- Input
  - One line with a string given from the standard input
  - The string consists of only `(`, `)`, `{`, `}`, `<`, `>`, `[`, and `]`, and there is no whitespace
  - The length of a string is no more than 100
- Output
  - Print the maximum depth if the brackets are well balanced
  - Print "`invalid`" if the brackers are not balanced.
- Example

```
[]{()<()>}([])
```

<Input 1>

```
3
```

<Output 1>

```
[][[()]{<(){}>}]
```

<Input 2>

```
4
```

<Output 2>

```
([]<{)>})
```

<Input 3>

```
invalid
```

<Output 3>