CPSC 331 — Fall 2013

Assignment #2 — Implementation and Application of a Stack

1 About This Exercise

This assignment concerns material covered in lectures up to and including the lecture of Monday, October 7, and and used to solve problems in tutorials up to Thursday, October 10.

This assignment can be completed by groups of up to three students and it is due by 11:59pm on Friday, October 25. Please see the main "Assignments" course page for information about requirements for the reports and code required for assignments as well about how to submit your assignment using Blackboard.

2 The Problems To Be Solved

1. Write the source code `ListStack.java` for a class that provides a singly linked-list implementation of a Stack. An implementation of a singly linked list is available and should be used for this, and your class should implement the Stack interface that has also been provided.

   Your class should be part of the package `cpsc331.assignment2`.

2. Consider an alphabet Σ that includes the lower case letters `a`–`z`, the upper case letters `A`–`Z`, the digits `0`–`9`, and the brackets "(", ")", "[" and "]".

   We will say that a string of symbols in this alphabet is **properly nested** if the left and right brackets are arranged in the way you would generally expect to see them. A *recursive definition* of the **set of properly nested strings** is as follows.

   (a) The empty string is a properly nested string.

   (b) Any string of symbols in Σ with length one that does not include any of "(," "),", "[" or "]" is a properly nest string.

   (c) If μ and ν are both properly nested strings of symbols (in the alphabet Σ) then so is the concatenation $\mu\nu$ of these strings.

   (d) If μ is a properly nested string of symbols (in the alphabet Σ) then so is (μ).

(e) If μ is a properly nested string of symbols (in the alphabet Σ) then so is $[\mu]$.

A computational problem concerning these strings can now be defined.

Properly Nested String Recognition

Precondition: A string ω of symbols over the alphabet Σ is given as input.

Postcondition: If ω is a properly nested string then the value `true` is returned as output. The value `false` is returned as output, otherwise.

This problem can be solved using an algorithm that makes use of a *stack* that stores symbols in Σ. The stack should be initialized to be empty. Using a `while` loop, the algorithm should consider each of the symbols in ω, in order from left to right.

- If a symbol is a left bracket — that is, either "(" or "[" — then this should be pushed onto the stack.
- If a symbol is a right bracket — that is, either ")" or "]" — then the stack should be examined and, if the stack is not already empty, a symbol should be popped off it:
  - If the stack is already empty then the algorithm should return `false`.
  - If the stack is not already empty, but the element that has been popped off the stack does not match it — that is, the element removed from the stack is "(" and the symbol in ω being processed is "]," or the element removed from the stack is "[" and the symbol in ω being processed is ")" — then the algorithm should return `false.`
  - Otherwise either the symbol removed from the stack is "(" and the symbol in ω being processed is ")" or the symbol being removed from the stack is "[" and the symbol in ω being processed is "]." In either case, processing should continue.
- Finally, if the symbol from ω is not a bracket (that is, not one of "(," ")," "[" or "]" at all) then processing should continue without examining or changing the stack.

If all of the symbols in ω have been processed without `false` being retuned then `true` should be returned if the stack is now empty and `false` should be returned, otherwise.

Write reasonably detailed pseudocode for an algorithm `isNested` that solves the "Properly Nested String Recognition" problem, with a stack, using the process that is given above. This should not be hard! You *do not* necessarily need to give a loop invariant for the `while` loop in this algorithm when you do this, but a bound function for it *should* be given. More information about this can be found in the notes that follow Question #3.

Suggestion: It will probably be quite helpful for you to have a better idea about what this algorithm is doing before you continue with the rest of the assignment. I suggest that you trace the execution of the algorithm on the following strings. Please do not include your traces of execution in your submission if you do this:

- [`a`] — a string that *is* properly nested
- `bc` — another string that *is* properly nested
- [(`a`)] — another string that *is* properly nested
- [ ](`a`[`b`]) — another strong that *is* properly nested
- ((`a`) — a string that *is not* properly nested
- [(`a`]) — another string that *is not* properly nested
- (`a`)) — another string that *is not* properly nested
-)`a`(— another string that *is not* properly nested.

3. Prove that each of the follow claims is correct.

   (a) If ω is a string of symbols in Σ that is properly nested and the algorithm `isNested` is executed with ω as input then this execution of the algorithm halts and `true` is returned as output.

   (b) If ω is a string of symbols in Σ and an execution of the algorithm `isNested` eventually halts, with `true` returned as output, then ω is correct.

   (c) The `isNested` algorithm correctly solves the "Properly Nested String Recognition" problem.

   Notes: There are at least two ways to solve parts (a) and (b). One way is to give a *loop invariant* (as well as a bound function) for the `while` loop in your algorithm, prove that this loop invariant is correct using the methods that have been described in this course, and use it to establish the partial correctness of the algorithm.

   The other is to follow a process resembling the one used in the tutorial exercise on stacks that involved using the recursive definition of the set of strings being recognized.

   I think that the *second* way is probably easier, for this problem, because it is actually quite tricky to give a loop invariant for this loop that is both correct and complete enough to be used to prove the partial correctness of this algorithm.

   A small number of **bonus marks** will be given, if you do not use a loop invariant to complete Question #3 but you give one anyway, and this loop invariant is both correct and complete enough to prove the partial correctness of this algorithm.

If you *do not* use or provide a loop invariant then you may **skip over the rest of these notes about Queston #3** and continue to Question #4.

With that noted, here is some additional terminology that might be useful if you are trying to give a correct and complete loop invariant for the `while` loop in your algorithm: If ω is a properly nested expression then there is a unique, well-defined **matching right bracket** for every left bracket in ω, and every right bracket *is* the "matching right bracket" for exactly one left bracket that appears before it in ω:

- The empty string and properly nested strings with length one do not include any brackets at all.
- If μ and ν are properly nested strings then the matching right bracket for any left bracket in $\mu\nu$ is the same as its matching right bracket in μ. That is, the positions of both the left bracket and its matching right bracket are the same in $\mu\nu$ as they are in μ.

  Similarly, if a left bracket and its matching right bracket have positions i and j in ν, respectively, and the string μ has length m, then this left bracket and its matching right bracket have positions $m+i$ and $m+j$, respectively, in $\mu\nu$.
- If μ is a properly nested string, and a left bracket and its matching right bracket have positions i and j, respectively, in μ, then the corresponding left bracket and the right bracket that matches it have positions $i+1$ and $j+1$, respectively, in both of the strings (μ) and $[\mu]$. The matching right bracket for the initial left bracket in each of (μ) and $[\mu]$ is the right bracket at the very end of the string.

A complete and correct loop invariant for this algorithm will almost certainly need to refer to both "matching right brackets" and to *prefixes* of strings.

4. Now write a Java program `CheckBrackets` —that is part of the package

```
cpsc331.assignment2
```

— to implement the algorithm you gave when you answered Question #2. It should be executed with zero or more character strings (separated by one or more blank spaces) also given on the command line, and it should display either the message

```
The input string is properly nested.
```

or the message

```
The input string is not properly nested.
```

or the message

```
Sorry!  You must provide at most one character string as input.
```

In particular, if no strings are given as inputs on the command line then the input should be interpreted to be the empty string, so that the message

```
The input string is properly nested.
```

should be displayed.

If a single input string is given as input and this is a string of symbols in Σ then the output message should be

```
The input string is properly nested.
```

if the input string really *is* a properly nested string, and the output message should be

```
The input string is not properly nested.
```

if it is *not* properly nested.

It does not matter what the program does if a single character string is given as input but this is not a string of symbols in Σ.

If two or more strings are given as input then the output message

```
Sorry!  You must provide at most one character string as input.
```

should be displayed.

A few sample runs of the program should therefore look like the following

```
> java cpsc331.assignment2.CheckBrackets
> The input string is properly nested.

> java cpsc331.assignment2.CheckBrackets [(a)]b
> The input string is properly nested.

> java cpsc331.assignment2.CheckBrackets ((a]
> The input string is not properly nested.

> java cpsc331.assignment2.CheckBrackets madam (a) [[
> Sorry!  You must provide at most one character string as input.
```

Please *do not* put quotation marks around the character strings on the command line.

Your program should include (at least) two methods as described below.

- The main method should check whether zero, one, or more than one string has been provided as input. It should pass the method that is described next the empty string, as input, if no input strings were received. It should pass the input string to this method as its input if one input string is received. It should not call this method at all, but simply print the required output string, if two or more strings are given as input.

  When either either no string or one string has been given as input, the main method should print

  ```
  The input string is properly nested.
  ```

  if the method described next returns `true`, and it should print

  ```
  The input string is not properly nested.
  ```

  if the method described next returns `false`.

- A method with signature

  ```
  public static boolean isNested (String w)
  ```

  should implement the algorithm that you described when answering Question #2. It should not bother to try to check whether its input string really *does* only include symbols in Σ, so it should not be necessary for this method to throw any exceptions at all.

5. Finally, design a suite of tests for the method `isNested` in the program that you wrote, when answering Question #4, and write a program `TestCheckBrackets`, that is also in the package

   ```
   cpsc331.assignment2
   ```

   and that can be used with `junit` to carry out the tests you designed. That is, it should be possible to carry out these tests by executing the command

   ```
   java org.junit.runner.JUnitCore cpsc331.assignment2.TestCheckBrackets
   ```

   after your program has been compiled.

   Your answer to this question should include a reasonably brief report explaining *why* you included each of the tests that you did. For full marks your testing should be reasonably thorough — that is, it should follow the guidelines for the design of black box tests and white box tests that were described earlier in the course.

Please be kind to the marker: Structure your report and your testing program in a way that makes it as easy to refer back and forth in them as possible.