version of *November 21, 2014 – 21 : 13*

Background

In this document, the word *binary tree* will either mean

- a binary *search tree* whose nodes are labelled by strings, or
- a *full* binary tree: nodes are labelled by strings, with every interior node having two children, otherwise arbitrary

Everything we say below applies to trees labelled with any data type with an ordering on it but we'll use strings for concreteness.

Fact 1. If T is a binary *search* tree, then the preorder traversal of T completely determines T. That is, you can draw T once you know *pre*(T). Put another way, there is only one binary search tree with a given preorder. *Convince yourself of this!*[1]

Fact 2. If T is a binary search tree, then the inorder traversal of T does *not* determine T. Put another way, there can be more than one tree with a given inorder traversal. *Convince yourself of this,* by drawing two different binary search trees with the same inorder traversal. (This is super-easy.)

Fact 3. If T is a *full* binary tree, not necessarily a binary search tree, then the preorder traversal of T does *not* determine T. *Convince yourself of this,* by by drawing two different binary trees with the same preorder traversal.

Fact 4. If T is a *full* binary tree, even if not a binary search tree, then the preorder traversal of T together with the inorder traversal of T *do* together determine T. That is, you can draw T once you know both *pre*(T) and *in*(T). *Convince yourself of this!*

Fact 5. If T is a *full* binary tree, even if not a binary search tree, then the preorder traversal of T together with the postorder traversal of T *do* together determine T. That is, you can draw T once you know both *pre*(T) and *post*(T). *Convince yourself of this!*

Summary If you have come to believe Fact 1 and Fact 4 and Fact 5, because you thought hard about them, then you figured out, at some intuitive level, an *algorithm* for deciding what T is, given

- the preorder (if we assume T is a binary search tree) or given
- the preorder together with the inorder (if we assume T is full) or given
- the preorder together with the postorder (if we assume T is full)

In this assignment you will implement those algorithms.

[1] By the way, once you believe this it should be obvious that if T is a binary search tree, then the postorder traversal of T also completely determines T, by symmetry.

Assignment

Represent traversals of trees as arrays of strings.

Write three static methods with the following specifications

1. `search_pre_to_post`:

   Takes as input an array purportedly representing the preorder traversal of some binary search tree T.

   Returns as output an array representing the postorder traversal of T.

   If the given input array cannot possibly be the preorder traversal of any binary search tree, write a message to that effect to standard output and return the empty array.

2. `pre_in_to_post`

   Takes as input two arrays, representing the preorder traversal and inorder traversal (in that order) of some full binary tree T.

   Returns as output an array representing the postorder traversal of T.

   If the given input arrays cannot possibly be the preorder and inorder traversals of any full binary tree, write a message to that effect to standard output and return the empty array.

3. `pre_post_to_in`

   Takes as input two arrays, representing the preorder traversal and postorder traversal (in that order) of some full binary tree T.

   Returns as output an array representing the inorder traversal of T.

   If the given input arrays cannot possibly be the preorder and postorder traversals of any full binary tree, write a message to that effect to standard output and return the empty array.

You will be given a file of starter code whose main method reads from standard input, calls your methods, and writes the results to standard out.

- Each line of input (hence, each tree) will have at most 100 strings.
- Input can be terminated either by a blank line or by the end-of-file character.
- If one line of strings are read then main should call `search_pre_to_post`.
- If two lines of strings are read then main should call `pre_in_to_post` and then call `pre_post_to_in`. This will generate two, probably different, answers.

An input line of lines could be erroneous input for any of several reasons:

- it might be a line of strings that cannot be the preorder traversal of any search tree. Your code should notice that, **print a message,** and **return the empty list** as result.
- it might be two lines that cannot be jointly a pair of the traversal of a full tree (this includes the possibility that the lists have different lengths). Your code should notice that, **print a message,** and **return the empty list** as result.

Strategy

This might seem daunting at first, but it isn't, really. There is *very little* code to write, and once you get the hang of it, it will be pretty straightforward once yo embrace thinking recursively. Once you have nailed this, you will be an awesome recursive programmer.

Sample input/output

- `search_pre_to_post`

  input `h b d e i j` will return output

  `e d b j i h`

- `pre_in_to_post`

  input `a b d e c` and `d e  b a c` will return output

  `bad input:  no full tree has these as preorder and inorder`

- `pre_post_to_in`

  input `a b c` and `b c a` will return output

  `b a c`

- `pre_post_to_in`

  input `a b d e c f g` and `d e b f g c a` will return output

  `d b e a f c g`

Grading Rubric

Since software development is not the primary subject matter of the class, the project grading will stress correctness. We continue to expect the good software development habits developed in previous classes, but in this class we will not ask you to explicitly present evidence of those habits for grading. For example, we assume that you have done thoughtful testing but showing us that test data is not part of your submission.

The project will be graded on a 20-point scale

- 5 pts documentation and structure:
  - Put the names of all students for whom the assignment is submitted at the beginning of each source code file
  - Annotate every method with a brief comment explaining what the method does.
  - Use newlines and indentation to produce clean, well-formatted code (we actually read your code, we don't just run it).

- 15 pts correctness:

  Here, ”correctness” means (i) getting the right answer and (ii) using algorithms with the right runtime. For this assignment this comes down to using recursion properly rather than trying some mad scramble to cover cases exhaustively. I actually don’t think it is possible to get these methods right except by programing recursively.