in calculus is so sound that there are no exceptions in real life; the exceptions are all unrealistic counterexamples that only a mathematician would figure out. Consequently, the practitioner is free to use the rule without concern about its validity, and without being in touch with its proof and the exact reasons as to why it works.

With algorithms, the usual state of affairs is just the opposite. When a problem definition is changed in even some seemingly innocuous way, the algorithm is likely to be inapplicable. If we are lucky, we may be able to adapt the algorithm to solve the new problem, or adapt the problem to be solved by the original algorithm. But even these simple changes may require the user to be very much in touch with the correctness arguments and the individual computations for the specific algorithm. Those in the business of applying algorithms are not free to forget the fundamentals. Sometimes a simple constraint can transform a problem from one that is easy to solve into one that is intractable. Sometimes it is easier to adapt an algorithm to a new application than to show why the adaptation is correct—and the consequence can be an erroneous solution. When solving algorithms problems, we must be on guard at all times.

1.8 Exercises

Ex. 1.1 Towers of Hanoi

Try to trace the Towers of Hanoi problem for a stack of five rings. If you get lost, don’t worry, Chapter 2 gives a systematic approach to trace such code as a way to give, hopefully, a better understanding of recursion, and its value as a tool and a concept.

Ex. 1.2 Towers of Hanoi

Devise a recursive solution for the Super Towers of Hanoi problem, where the rings are initially distributed arbitrarily among the three poles but are, of course, stacked legally on each pole, so that higher rings have smaller diameters. The rings still have the names $1, 2, \ldots, n$, where ring 1 is the smallest.

Suggestion: How about defining

$$STH(n, A, B, C)$$

where just as in the regular Towers of Hanoi solver. In addition to the move function used in the TowersOfHanoi procedure of Section 1.4, use the predicate IsItThere($j, X$), which is true if pole $X$ contains the ring named $j$, and is false otherwise.

The code will have cases because you cannot know in advance where a particular ring is located.

a) How would a beginner try to solve this problem?

b) Suppose that Radio Shack sold a generalized Towers of Hanoi supersolver to solve this problem for $n - 1$ rings. How would you use this supersolver to solve the problem for $n$ rings?

c) Present the general solution.

Ex. 1.3 Towers of Hanoi

Suppose you have a Towers of Hanoi problem with four poles: $A$, $B$, $C$, and $D$. All $n$ rings initially sit on pole $A$. Poles $A$, $B$, and $C$ can hold all $n$ rings, but pole $D$ can hold just 1 ring.
Your task is to move all $n$ rings, subject to the usual rules of the game, to pole $B$ in as efficient a manner as possible.

a) What would a beginner do?

b) What would an advanced Towers of Hanoi problem solver think about?

c) What is the best solution to the problem? Think about the question yourself for a while. Then read the hint.

Hint: You put the solver to good use?

Suppose you had a Radio Shack Solver for this problem for $n$ minus two rings. How would you

Ex. 1.4  Towers of Hanoi

The purpose of this problem is to point out that not all recursion is the same. In particular, some problems have recursive subproblems that are (somewhat) different from the original problem. For example, the constraint that makes the following problem different from the standard Towers of Hanoi problem does not have to be satisfied by its recursive subproblems. Likewise, some recursive algorithms issue calls to other recursive algorithms.

For this problem, all of the standard $TH$ rules apply, all $n$ rings initially sit on pole $A$, there are the three poles $A$, $B$, and $C$, and the objective is again to move the stack to pole $B$. The key difference is that the rings have top and bottom surfaces with different colors. Each ring initially begins on $A$ with its top side colored red, and the bottom side colored white. Whenever a ring is moved from one pole to another, it is flipped over, which reverses the color on the top of the ring.

a) What are the colors of the top sides of the rings on stack $B$ when the standard $TH$ solver (with each ring flipped over whenever it is moved) is used to move the rings to $B$? Very big hint: Think about the question yourself for a while. Then read the hint.

Hint: Look at the $TH$ pseudocode. How many times is the full set of $n-1$ rings moved? Although the names of the poles change in the two subroutine calls, the number of moves a specific ring is relocated does not.

b) Present an efficient $RTH$ solver that moves all rings to $B$ in a way where the top sides are all red. Hint: You must understand a before you can tackle b.

Answer to a) The Towers of Hanoi code moves the collective entity of $n-1$ rings twice. The names of the poles change in the two subroutine calls, the number of moves a specific ring is relocated does not.

Ex. 1.5  Towers of Hanoi, modeling

The Circular Towers of Hanoi ($CTH$) problem is exactly the same as the regular Towers of Hanoi problem but with one extra constraint: Any single move can only move a ring one pole to the right, so that a move from pole $A$ must go to pole $B$, provided the move is legal for the unrestricted problem. Likewise a move from $B$ must be to $C$ and a move from $C$ must be to $A$. 
Present an efficient solution to the $CTH$ problem, which is to move all of the rings from $A$ to $B$.

Comments: One of the interesting parts of the problem is to understand why the regular $TH$ solver does not give a correct answer. The other challenge is to devise a formulation where you can be sure that the ring moves satisfy all of the constraints.

Hint: Moving the stack from $A$ to $C$ is not algorithmically equivalent to moving the stack from $A$ to $B$ (because the $A$-to-$B$ moves is between consecutive columns whereas the $A$-to-$C$ move is not). So you need two recursive coroutines. One does moves a stack one post, and one moves two posts. The moves of a single ring have to be done differently as well.