Recursion — Study Questions
The following is adapted from the course textbook "Data Abstraction and Problem Solving with C++" as well as Prof. Stewart Weiss' notes for CSCI 235 on Recursion and Problem solving

- **Approach:**
  - Determine the base case.
  - Does the base case perform and action or return a value?
  - Determine the recursive call (how is the problem getting smaller?)
  - What is the relationship between the base case ad the recursive call(s)?
  - How many recursive calls? If more than one, execute all or some?

- A DNA string, also called a DNA strand, is a finite sequence consisting of the four letters A, C, G, and T in any order. The four letters stand for the four nucleotides: adenine, cytosine, guanine, and thymine. Nucleotides, which are the molecular units from which DNA and RNA are composed, are also called bases. Each nucleotide has a complement among the four: A and T are complements, and C and G are complements. Complements are chemically-related in that when they are close to each other, they form hydrogen bonds between them. For example, the complement of TGGC is ACCG, and the complement of TCGA is AGCT. Notice that this last string has the property that its complement is the same as the string when read backward. A sequence of nucleotides is palindromic if the complement read right to left is the same as the string read from left to right. For example, the DNA string TGCAACGCGTTGCA is palindromic because the complement is ACGTTGCGCAACGT, which when read backwards is the original string.
  Write a recursive function that, when given a DNA string s, returns true or false depending on whether s is palindromic. Note that this is different from the standard definition of palindrome.

  Another use of recursion is to define infinite sets of various types. There are always at least two rules. The first is analogous to a base case in an induction proof, and can be called the basis clause or simply the basis. The second is the inductive clause. For example, the set of natural numbers, denoted N, can be define by the following recursive definition:

  1. $0 \in \mathbf{N}$.
  2. $n \in \mathbf{N} \Rightarrow n + 1 \in \mathbf{N}$

  Repeated application of Rule 2 generates the set of all natural numbers. Implicit in any definition of a set is that the set contains nothing but what the definition places into it. This does not have to be stated explicitly. The fact that 1.5 is not a natural number is because it is not placed into the set by either of Rules 1 or 2. Some authors make this rule explicit and call it the extremal clause.

Another set of numbers is defined by this recursive definition:

1. $0 \in A$
2. $n \in A \Rightarrow 2n + 1 \in A$

If you apply Rule 2 repeatedly, you will see that this set consists of the numbers 0, 1, 3, 7, 15, 31, and so on, which is the set $\{2n - 1 | n \in N\}$.

- These two examples show that recursive definitions generate the numbers that are in the set by repeated application of the rules.

- **Grammars:** A rule in a grammar is of the form

  ```
  variable = replacement_string
  ```

  which means that the variable on the left can be replaced by the replacement string on the right. The replacement string is a sequence of variables and/or symbols of the underlying alphabet. When a variable appears in the replacement string, it is enclosed in angle brackets (<>) to distinguish it from the non-variables in the replacement.
  Grammars usually have a designated start variable, which is the one that appears on the left-hand side of the rule and is the rst rule to be applied. Some books use a special letter such as S to denote this, but here it is enough to use a symbol that is self-explanatory.
  When a variable can be replaced by more than one replacement string, we use a vertical bar as a symbol meaning or.

  The language AnBn represents the string that consists of n consecutive As followed by n consecutive Bs. The grammar for this language is:

  ```
  <legal_word> = empty string | A <legal_word> B
  ```

  Write a recursive function that, when given a string s, returns true or false depending on whether s is in the language AnBn.