BigNum Class and Floating-Point Limitations
Cpt S 422 Homework Assignment
by Evan Olds

Assignment Instructions:

Read all instructions *carefully* before you write any code.

In this assignment you will implement the **BigNum** class, which represents a floating-point type that can grow (almost) unbounded, as long as there is enough system memory. This class will let you do many floating-point operations without rounding or truncation. It will also allow you to determine whether or not certain operations can be done with floats and doubles without data loss.

Implement the BigNum Class

Implement the BigNum class as an immutable type in the file BigNum.cs and in the namespace CS422. Store the number and perform operations on it in base-10, so that we can ensure the ability to exactly represent numbers like 9.4. Your implementation is allow to have certain technical limitations, such as using an integer for a power of 10, which would in theory limit you to numbers with about 2-billion-digits. But it must not have any limitations beyond that and should be able to grow as large as memory on the system allows. Include the following members in the class:

- public BigNum(string number)
  - Instantiates a BigNum from a real number string.
  - Validate the string the string and throw an exception if it does satisfy the criteria below
    - The string must start with a minus sign, numeric character, or decimal point
    - No whitespace is allowed
    - Null or empty strings are not allowed
    - The only allowed characters are
      - '-' (at most 1 and only ever valid as the very first character)
      - '.' (at most 1)
      - [0-9]
  - Throw an argument exception if the string doesn't satisfy these requirements
  - If the string is a valid number, this constructor will initialize the BigNum to an exactly equal value. There must not be any rounding or truncation.
- public BigNum(double value, bool useDoubleToString)
  - Constructs the BigNum from an existing double value. There are 3 cases.
  - If the double value is NaN, +infinity, or –infinity, then the BigNum is initialized to an undefined value, regardless of the value of the useDoubleToString parameter. This

undefined value special case is described more below in the IsUndefined property requirements.
    - Otherwise, if useDoubleToString is true, you convert the value to a string using ToString and construct the BigNum in the same fashion as the constructor that takes a string.
    - The last case is that value is a real number and useDoubleToString is false. In this case you initialize the BigNum to the ***exact*** value represented by the double. As we saw in class this is often different from what ToString would return for a double. You must parse through the bits in the value and interpret them according to the IEEE 754 specification for double precision floating point values.
- public override string ToString()
    - Returns a string for the base-10 representation of this number.
    - Implement this to use the minimum number of digits needed to accurately represent the number, without scientific notation.
    - There must not be any whitespace characters in the string.
    - If the number is an integer, do not include a decimal point at all.
    - If the absolute value of the number is < 1, do not include any numeric digits before the decimal point. Zero is a special case in which case you just return "0".
    - Include the '-' as the first character for all negative numbers, but do NOT precede positive numbers with a '+'.
    - Do not include unnecessary leading or trailing '0' characters. For example, if the value is 0.4, the only string representation that is valid given the previously mentioned rules is ".4". Values of "0.4", "0.40", ".40", although they technically represent the same number, will not be accepted for credit.
    - For the special case of the number being undefined (described more below), just return the string "undefined".
    - This is one of the most important functions, as it provides a lossless representation of the number. Since there is no scientific notation or rounding, all relevant digits are displayed and you will be able to go back and forth between BigNums and strings with no data loss from the conversions.
- public bool IsUndefined { get; }
    - Have your BigNum support a special-case state where it is undefined. This is similar to the NaN and infinity cases for floating-point values, but we're bundling all special cases where it doesn't represent a real number into one state.
    - You can manage this in a variety of ways, but the simplest recommendation is just to have a bool member variable that is true when the number is undefined and false when it is not.
    - For all binary operators listed below (+ - * /), return undefined if either operand is undefined.
- public static BigNum operator+(BigNum lhs, BigNum rhs)
    - Implements addition as a **lossless operation**
- public static BigNum operator-(BigNum lhs, BigNum rhs)
    - Implements subtraction as a **lossless operation**
- public static BigNum operator*(BigNum lhs, BigNum rhs)

  - Implements multiplication as a **lossless operation**
  - Do NOT implement this by starting at 0 and adding rhs repeatedly lhs number of times (or adding lhs repeatedly rhs number of times). This implementation is far too slow to be reasonable and will not be accepted for any credit. Plus, it wouldn't work if neither operand were an integer. You need to come up with a way to do this operation in worst case O(n), where n is the combined number of digits in the two numbers.
- public static BigNum operator/(BigNum lhs, BigNum rhs)
  - Implements division as a potentially lossy operation
  - Return an undefined BigNum if either lhs or rhs is undefined, or if rhs is zero.
  - Have an implementation that does a finite number of iterations and gets a reasonable number of digits before terminating further computation. Have at least 20 digits by default, but make sure you do not go on forever because the operation may not be possible to compute exactly.
  - Like the multiplication, you cannot do an implementation that loops 2 billion times if one of the arguments is 2 billion. You need an O(n) solution, where n is the combined number of digits in the two numbers.
- public static bool operator>(BigNum lhs, BigNum rhs)
- public static bool operator>=(BigNum lhs, BigNum rhs)
- public static bool operator<(BigNum lhs, BigNum rhs)
- public static bool operator>=(BigNum lhs, BigNum rhs)
- public static bool IsToStringCorrect(double value)
  - Utility function that determines whether or not ToString for the specified value generates an exact representation of the stored value.