CSE 3100 Systems Programming	Out: 09/15/16
Assignment #2	Due: 09/21/16

For instructions on how to checkout the template code for the assignment and submit solutions using git, see http://dna.engr.uconn.edu/moodle/mod/page/view.php?id=129

Exercise 1. ***Polynomial evaluation (70 points)***

(a) Write a recursive function **double power(double x, int n)** *based on the following identity:*

$$x^n = \begin{cases} x^{\lfloor n/2 \rfloor} \times x^{\lfloor n/2 \rfloor} \text{ if } n \text{ is even} \\ x \times x^{\lfloor n/2 \rfloor} \times x^{\lfloor n/2 \rfloor} \text{ if } n \text{ is odd} \end{cases}$$

Make sure to use an appropriate base case to avoid infinite recursion!

(b) A univariate polynomial

$$a_0 + a_1x^1 + a_2x^2 \ldots + a_{n-1}x^{n-1} + a_nx^n$$

is a mathematical expression involving the sum of powers of a variable x multiplied by constant coefficients. Each term in the above sum is called a monomial, *a_i, $i = 0, \ldots, n$ are called the* coefficients *of the polynomial, and n, the largest power n with non-zero coefficient, is called the* degree.

A simple representation of a polynomial uses an array for storing the coefficients. Complete the provided code by implementing a function that returns the value of a given polynomial for a given value of x. Write two versions of the function. The first version should use the power function from part (a). The second version should implement ***Horner's rule****, which reduces the number of necessary multiplications by factoring out powers of x as follows:*

$$a_0 + x\left(a_1 + x\left(a_2 + x\left(\ldots x\left(a_{n-1} + xa_n\right)\ldots\right)\right)\right)$$

The main function in the provided template reads from the standard input a double x, a non-negative degree n, and the $n+1$ integer coefficients of a polynomial, then prints the values computed by each of the two versions of your evaluation function.

Exercise 2. ***Super-permutations (30 points)***

A permutation *of the integers from* 1 *to n is an ordering of these integers. A natural way to represent a permutation is to list the integers in the desired order. For example, for $n = 5$, a permutation might look like 2, 3, 4, 5, 1. However, there is alternative way of representing a permutation: you create a list of n numbers where the i-th number is the position of integer i in the permutation. For the above permutation this alternative representation is 5, 1, 2, 3, 4 since 1 appears on position 5, 2 appears on position 1, and so on (note that in math position numbering starts from 1). A* super-permutation *is a permutation for which the two representations are identical. For example, 1, 4, 3, 2 is a super-permutation. You have to write a program which detects if a given permutation is a super-permutation or not.*

Input: *The program should read from the standard input one or more test cases. The first line of each test case contains an integer n ($1 \leq n \leq 100{,}000$). Then a permutation of the integers 1 to n follows on the next line. There is exactly one space character between consecutive integers. You may assume that every integer between 1 and n appears exactly once in the permutation. The last test case is followed by a zero.*

Output: *For each test case print to the standard output one line with the answer formatted as in the sample below.*

Sample Input:

```
4
1 4 3 2
5
2 3 4 5 1
1
1
0
```

Sample Output:

```
super
not super
super
```