You are to write a program to compute the derivative of a polynomial. For a polynomial
\[ p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0, \]
the derivative of \( p \) is defined to be
\[ p'(x) = n a_n x^{n-1} + (n-1) a_{n-1} x^{n-2} + \cdots + a_1. \]
You may assume that the degree of \( p \) is at least 1 and no more than 10, and that the coefficient on the highest-degree term is nonzero. You may also assume that all coefficients are integers. Note that even though the constant term \( a_0 \) is not used in the computation of the derivative, it still must be included on the input. The output polynomial must be formatted exactly as shown in the examples.

Example 1:

Enter degree: 4
Enter coefficient for \( x^4 \): 1
Enter coefficient for \( x^3 \): 2
Enter coefficient for \( x^2 \): 3
Enter coefficient for \( x^1 \): 4
Enter coefficient for \( x^0 \): 5

\[ 4x^3 + 6x^2 + 6x^1 + 4 \]

Example 2:

Enter degree: 5
Enter coefficient for \( x^5 \): 5
Enter coefficient for \( x^4 \): -4
Enter coefficient for \( x^3 \): -2
Enter coefficient for \( x^2 \): 3
Enter coefficient for \( x^1 \): 0
Enter coefficient for \( x^0 \): 0

\[ 25x^4 - 16x^3 - 6x^2 + 6x^1 + 0 \]
A string \( s \) is a subsequence of a string \( t \) if \( s \) can be obtained from \( t \) by removing 0 or more characters. Write a program that takes as input a string \( t \) and a string \( s \) and reports whether \( s \) is a subsequence of \( t \). You may assume that each string has length at most 80.

**Example 1:**

Enter t: facetious
Enter s: aeiou

aeiou is a subsequence of facetious.

**Example 2:**

Enter t: education
Enter s: aeiou

aeiou is not a subsequence of education.
Newton’s method may be used to compute the square root of a real number \( x \) as follows. An initial guess \( y_0 \) is made. Given \( y_k \), the next approximation \( y_{k+1} \) is computed using the formula,

\[
y_{k+1} = \frac{y_k + \frac{x}{y_k}}{2}.
\]

Write a program that takes as input a value for \( x \), an initial guess \( y_0 \), and a number of iterations \( k \), and produces as output the value \( y_k \) produced after \( k \) iterations and the value \( y_k^2 \). You may assume that all inputs are at least 1, and that \( k \) is an integer. While it is possible that this method could create intermediate results that exceed the capacity of a standard floating-point value, you do not need to handle this case (the examples below and the test cases should not cause this behavior). Your results may differ slightly from those shown below, but should agree to at least 3 significant digits.

**Example 1:**

Enter \( x \): 65536  
Enter initial guess: 10  
Enter number of iterations: 10

The square root of 65536.0 is 256.0  
256.0^2 = 65536.0

**Example 2:**

Enter \( x \): 2.56  
Enter initial guess: 1.1  
Enter number of iterations: 3

The square root of 2.56 is 1.6000044  
1.6000044^2 = 2.5600142
Write a program that takes as input an integer $n$ and integers $a_1, a_2, \ldots, a_n$, and produces the coefficients of the $n$th-degree polynomial representing the product,

$$(x - a_1)(x - a_2) \cdots (x - a_n).$$

You may assume that $1 \leq n \leq 10$. You must format your output polynomial exactly as shown in the examples.

**Example 1:**

Enter number of terms: 4
Enter a1: 1
Enter a2: -1
Enter a3: 2
Enter a4: -3

\[1x^4 + 1x^3 + -7x^2 + -1x^1 + 6\]

**Example 2:**

Enter number of terms: 5
Enter a1: 0
Enter a2: 0
Enter a3: 1
Enter a4: 1
Enter a5: 2

\[1x^5 + -4x^4 + 5x^3 + -2x^2 + 0x^1 + 0\]
You are to write a program which takes as input a date in the Gregorian calendar (the one used in the U.S.) and translate it to the corresponding date in the Indian calendar (the one used in India). The following rules apply:

- Year $n$ in the Indian calendar begins on the 81st day of year $n + 78$ in the Gregorian calendar. This is March 22 in a non-leap year, or March 21 in a leap year.
- Year $n$ in the Indian calendar is a leap year if year $n + 78$ in the Gregorian calendar is a leap year. Thus, if $n + 78$ is divisible by 4 but not 100, or if $n + 78$ is divisible by 400, then year $n$ is a leap year in the Indian calendar.
- In a non-leap year, months 2, 3, 4, 5, and 6 in the Indian calendar each have 31 days, and the remaining months each have 30 days. In a leap year, an extra day is added to the first month.
- In the Gregorian calendar, months 4, 6, 9, and 11 each have 30 days, month 2 has 28 days (29 in a leap year), and the remainder each have 31 days.

You may assume the input is a valid date in the Gregorian calendar, no earlier than 1800. You must represent all dates in input and output numerically, using 1-12 to indicate the month.

**Example 1:**

Enter month: 3
Enter day: 20
Enter year: 2000

Month: 12
Day: 30
Year: 1921

**Example 2:**

Enter month: 6
Enter day: 22
Enter year: 2002

Month: 4
Day: 1
Year: 1924
Newton’s method may be used to compute the \( n \)th root of a real number \( x \) as follows. An initial guess \( y_0 \) is made. Given \( y_k \), the next approximation \( y_{k+1} \) is computed using the formula,

\[
y_{k+1} = \frac{(n - 1)y_k + \frac{x}{y_k^{n-1}}}{n}.
\]

Write a program that takes as input values for \( x \) and \( n \), an initial guess \( y_0 \), and a number of iterations \( k \), and produces as output the value \( y_k \) produced after \( k \) iterations and the value \( y_k^n \). You may assume that all inputs are at least 1, and that \( n \) and \( k \) are integers. While it is possible that this method could create intermediate results that exceed the capacity of a standard floating-point value, you do not need to handle this case (the examples below and the test cases should not cause this behavior). Your results may differ slightly from those shown below, but should agree to at least 3 significant digits.

Example 1:

Enter x: 65536
Enter n: 4
Enter initial guess: 10
Enter number of iterations: 10

The 4th root of 65536.0 is 16.0
16.0^4 = 65536.0

Example 2:

Enter x: 2.56
Enter n: 2
Enter initial guess: 1.1
Enter number of iterations: 1

The 2th root of 2.56 is 1.7136364
1.7136364^2 = 2.9365497