1. Area of Polygon (10 Points)
The area of a regular polygon, a polygon having all $n$ sides of the same length, $s$, and all internal angles congruent, is...

$$A = \frac{1}{4} ns^2 \cot\left(\frac{\pi}{n}\right)$$

$\cot(\cdot)$ it the cotangent function, and is equivalent to the reciprocal of the tangent ($\tan(\cdot)$) of an angle.

Write a program that calculates the area of a regular polygon given the number of sides and the length of a side. Your program must ensure that the number of sides is at least 3 before performing any calculations.

2. Word Count (10 Points)
Count and display the number of words of a user-specified length in a user-entered string. Words are terminated by white space (space, tab, and new line) and punctuation (period, comma, colon, and semicolon). Punctuation may be assumed to always be immediately followed by whitespace. This implies a punctuation mark may be assumed to never immediately follow another punctuation mark.

For example, consider the following input.

*Word Length:* 7

*Sentence:* I studied from morning till evening.

Your program must indicate that there are 3 words of length 7 when given this input.

3. Relative Humidity (10 Points)
Calculate the relative humidity\(^1\) (the percentage of water vapor that the air holds relative to the maximum that it can hold at a particular temperature) given the following parameters by the user.

- $T_F$ – temperature (input in degrees Fahrenheit)
- $T_{DF}$ – dew point (input in degrees Fahrenheit)

You will need to convert these temperatures to Kelvin to use the formulas farther below.

- $K = (F-32)(5/9)+273.15$

The following values are constants and should be declared and used as such in your program.

- $e_{so}$ – reference saturation vapor pressure (6.11 hPa\(^2\))
- $T_0$ – reference temperature (273.15 K)
- $l_v$ – latent heat of vaporization of water ($2.5 \cdot 10^6$ J / kg)
- $R_v$ – gas constant for water vapor (461.5 J · K / kg)

The following are intermediate values that you will calculate.

- $e$ (vapor pressure) = $e_{so} \cdot \exp( l_v/R_v \cdot (1/T_0 - 1/T_d))$
- $e_s$ (saturation vapor pressure) = $e_{so} \cdot \exp( l_v/R_v \cdot (1/T_0 - 1/T))$

And, the following is your output.

- $RH = e/e_s \cdot (100\%)$

\(^1\) Relative humidity is discussed and these formulas are presented at http://www.faqs.org/faqs/meteorology/temp-dewpoint/.

\(^2\) The "h" represents "hecto-" or 100. The hPa (hectopascal) is a commonly used unit in atmospheric pressure calculations.
4. Statistics (20 Points)

Write a program to calculate statistics of a list of real numbers entered by the user. First, ask the user how many numbers are in his or her list (there must be at least 1). Then, prompt the user to input the values.

Calculate and output the **mean**, **median**, and **standard deviation** of the list of values. The mean is the average. For a list of odd length, the median is the number from the list such that half of the remaining numbers are larger (or equal) and the other half are smaller (or equal). For a list of even length, the median is the average of the two numbers from the list such that half of the remaining numbers are larger than (or equal to) each of the two numbers and the other half are smaller than (or equal to) each of the two numbers. The standard deviation is given by \( \sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_i - \mu)^2} \), where \( N \) is the length of the list, \( x_i \) are the elements of the list, and \( \mu \) is the mean.

For example, if the set of numbers is \{8, 3, 7\}, the median is 7 since half of the 2 remaining numbers are larger than 7 and half are less than 7. “7” is the number that would be in the middle if you sorted the list. Consider the list \{1, 9, 3, 6\}. In this case, the sorted list is \{1, 3, 6, 9\}. Since the length is even, we take the average of the middle 2 numbers to get the median, \((3 + 6) / 2 = 4.5\).

5. Least Common Multiple (20 Points)

Calculate and display the least common multiple (LCM) based on user input of two positive integers. You may assume valid input. You must provide correct output for inputs up to \( 2^{15} \cdot 1 = 32,767 \). The least common multiple is the smallest number such that each member of a set of numbers (your two inputs in this case) divides into it without leaving a remainder.

For the case of two numbers, if you know the greatest common divisor (GCD), the largest number that divides into each member of a set of numbers without leaving a remainder, you can calculate the LCM by taking the product of the two inputs and dividing it by the GCD.

The GCD of a set of two numbers can be found by using Euclid’s algorithm. Euclid’s algorithm maintains a set of two numbers, and iterates as long as neither of the numbers is 0. During an iteration, the larger number is replaced with the remainder of itself divided by the smaller number. The non-zero number remaining is the GCD. If the GCD is 1, we say that the numbers are “relatively prime” and the LCM is simply equal to the product of the numbers.

For example, the GCD of 100 and 36 can be found using Euclid’s algorithm, generating the following number pairs: \( (100, 36) \), \( (28, 36) \), \( (28, 8) \), \( (4, 8) \), \( (4, 0) \). In the second pair, the larger number, 100, is replaced with the remainder of 100 divided by 36 \((100 \div 36 = 2\) \( \cdot \) \( 36 + 28\)\). The iteration stops when one of the numbers becomes 0 at which time the other number is the GCD. Hence, \( (4, 0) \) tells us the GCD is 4. So, the LCM is \( 100 \cdot 36 / 4 = 900 \).
6. Modified Birthday Problem³ (20 Points)

Write a program to determine the minimum number of people that must be in a room such that the probability that two of them have a birthday within M days of each other (where M is input by the user and ranges from 0 to 14 — 0 indicates birthdays falling on the same day) is at least 50%.

Only approximate solutions to this problem are known. You will use a Monte Carlo simulation to simulate random events (the birthdays of people in the room in this case) and count the successful events (that there is a pair within no more than M days of each other (we’ll call this event E)). This approach is virtually guaranteed to give you the correct answer if you simulate each of the above events at least 10,000 times. Specifically, you will start with N=2 people and run 10,000 simulations (you can think of this as 10,000 different rooms). This will allow you to estimate the probability of E. You will then increase N by 1 and repeat until the probability is at least 50%. For our purposes, a birthday can be taken as a number from 0 to 364 (number of days after January 1 on which the birthday falls). Do not be concerned with leap years.

Note: Your program will need random integers to solve this using the Monte Carlo approach. See the end of this document for information on how to generate these.

7. Invoice Summary Report (40 Points)

You are given two files (you will take the filenames as input from the user). The first file contains information about items and their current prices. The second contains information about current orders, primarily how many of various items have been ordered. You must combine this information to determine and output the total value of each order along with the total value of all orders.

Each line of the items file contains the item ID number, the item description (all letters, no spaces), and the price in US dollars. For example, the file might look like...

```
17  Widget  2.50
23  Sprocket 0.23
35  Cog   3.49
```

The orders file contains two different types of lines. The first line type contains the word “ORDER” followed by an order number; it designates the beginning of an order. The second line type contains an item ID number followed by the item quantity. For example, the file might look like...

```
ORDER 53
23 23
35 14
ORDER 31
23 71
17 6
```

So, order 53 is for 23 sprockets (total value $5.29) and 14 cogs (total value $48.86) and has a grand total of $54.15. Similarly, order 31 is for 71 sprockets ($16.33) and 6 widgets ($15.00) for a grand total of $31.33. So, your output would be:

```
ORDER 53: $54.15
ORDER 31: $31.33
TOTAL:   $85.48
```

You are not required to align the results in any particular manner.

³The birthday problem is discussed at http://mathworld.wolfram.com/BirthdayProblem.html.
8. Prefix Calculator (40 Points)

"Prefix" notation is the formal term for writing arithmetic expressions such that the operators (addition, division, etc.) come before the operands (numbers, variables, expressions, etc.). Alternatives to prefix notation include "infix" notation, the way we normally write expressions, and "postfix" notation, where the operators come after the operands. Prefix notation is helpful in some computer applications where it is useful to first know which calculation is going to be performed so that one can prepare the hardware or other processing unit for that operation. Prefix notation has the added advantage that parentheses are never necessary.

For example, the following are prefix expressions shown with an equivalent infix expression and the corresponding value.

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Infix</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ 1 * 2 4 OR + * 2 4 1</td>
<td>1 + 2 * 4</td>
<td>9</td>
</tr>
<tr>
<td>− / 7 8 2</td>
<td>7 / 8 − 2</td>
<td>−1.125</td>
</tr>
<tr>
<td>+ * 7 ^ 2 2 3</td>
<td>3 + 2 ^ 2 * 7</td>
<td>31</td>
</tr>
<tr>
<td>+ 1 * 2 ^ 3 4</td>
<td>1 + 2 * 3 ^ 4</td>
<td>163</td>
</tr>
<tr>
<td>+ * 1 2 ^ 3 4</td>
<td>1 * 2 + 3 ^ 4</td>
<td>83</td>
</tr>
<tr>
<td>+ 3 * 2 − 1 4</td>
<td>(1 − 4) * 2 + 3</td>
<td>−3</td>
</tr>
</tbody>
</table>

Write a prefix calculator that accepts expressions consisting of floating point values, the operators +, −, *, /, and ^ with their common meanings, and produces the correct result. For example, if the user enters "− / 2 3 / 4 5.7, you will output "−0.0350877". The exact number of displayed figures is not important – do something reasonable (i.e., do not throw away the fractional part).

You may assume that the expressions are correctly formed (sequences of numbers and operators with one more number than operator, etc.) and will be entered on a single line. You may require either that there are none or (one or more) spaces between numbers and operators (although this assumption is not necessary, it makes certain designs easier to implement), but your program must tell the user about such a requirement.

Here is an algorithm that will evaluate a prefix expression. (You are not required to use this particular algorithm.)

- Create a stack that will contain values. A stack is a data structure like a stack of plates in which the item most recently added to the structure comes off first.
- Then, scan through the expression entered by the user from right to left (this can be accomplished by putting all input on a separate "token stack" as it is received), processing each token (number or operator) as you find it...
  - If the token is a number, put it on the value stack.
  - If the token is an operator...
    - Replace the two numbers on the top of the value stack with the result of the operator applied to the two numbers.
    - (If there are fewer than two numbers on the stack when you encounter an operator, the expression was not valid. You are not required to check for this type of error, however.)
- If you did everything correctly, you will end up with a stack with a single value, the answer, which must be reported to the user.

---

9. State Transition Matrixes and Markov Processes (40 Points)

We can describe the observed state of certain systems by category labels. For example, the system might be the Earth’s atmosphere, the observed state might be the weather at MSOE on a given day, and the category label might be one of “sunny”, “cloudy”, and “rainy”. For prediction purposes, we can predict tomorrow’s weather category based only upon today’s category. Given today’s category, we assign a fixed probability to each of the possible weather categories for tomorrow. This defines what is known as a first-order Markov process.

For example, let there be a 70% chance that a sunny day will be followed by another sunny day, a 20% chance that it will be followed by a cloudy day, and a 10% chance that it will be followed by a rainy day. Further, let there be a 30% chance that a cloudy day will be followed by a sunny day and a 20% chance that it will be followed by a rainy day. Then, there must be a 50% chance that a cloudy day will be followed by another cloudy day so that the probabilities sum to 100%. Finally, let there be a 60% chance that a rainy day is followed by a cloudy day and a 40% chance that it is followed by another rainy day. So, there is no chance, in our model, that a rainy day is followed by a sunny day.

All of the information in the above paragraph can be represented by the following matrix, called a state transition matrix, where the rows represent today’s state and the columns contain the probabilities of each possible state for tomorrow given the row. We assign “sunny” to the first row and column, “cloudy” to the second row and column, and “rainy” to the third row and column. Note that each row sums to 1 (or 100%) since the probabilities for each type of day must sum to 1.

\[
\begin{pmatrix}
0.7 & 0.2 & 0.1 \\
0.3 & 0.5 & 0.2 \\
0 & 0.5 & 0.4 \\
\end{pmatrix}
\]

Now, if it is sunny today and we want to know what the probability that it is cloudy 2 days from now, we need to sum three probabilities, one for each of the intermediate states. This is represented in the following diagram.

So, we want the sum of the probabilities of the 3 paths: $0.7 \cdot 0.2 + 0.2 \cdot 0.5 + 0.1 \cdot 0.6 = 0.14 + 0.10 + 0.06 = 0.30$. We multiplied the items from the first row of the matrix (representing transitions from the initial, sunny day) with the corresponding items from the second column of the matrix (representing transitions to the terminal, cloudy day) and summed them. This operation is called an “inner product.” By taking the inner product of each of the three rows, representing the initial states, with each of the three columns, representing the terminal states, we compute a total of 9 values that constitute the 2-day transition matrix, which we represent by MM or $M^2$ ($M$ squared). Each element in the result is the inner product of the corresponding row from the 1st matrix ($M$) with the corresponding column of the 2nd matrix (also $M$).

The 3-day transition matrix, $M^3$, can be computed as $M(MM)$ or $(MM)M$ since matrix multiplication is associative. (Matrix multiplication is not commutative, but that does not matter here since we are multiplying 3 identical matrices.)

Write a program that inputs the number of days, ensures it is between 1 and 31 before continuing, and computes and displays the corresponding state transition matrix for that number of days. Your program must use $M$ as given above, but must be written such that the 9 values of the matrix could be easily changed.
Generating Random Integers

C++
You will need to use a pseudorandom number generator to solve some problems. One such generator is part of the C standard library and is included in standard C++. This generator consists of two functions. To use these two functions you must `#include <cstdlib>.

The first of these functions, `void srand(int)`, “seeds” the random number generator. That is, it starts the generator at a specified point. Note that srand should only be called once in a program. So that the random number generator will usually start at a different point each time the program is run, you must seed it with a different value on each run. This can be accomplished by using the time, which changes each second. You can gain access to the standard C functions for working with time via `#include <ctime>`. The following line will use the current time to seed the random number generator... `srand(static_cast<unsigned int>(time(NULL)))`;

The other function, `int rand()`, returns an effectively random value. The function rand() returns integer values between 0 and RAND_MAX where RAND_MAX is a constant defined in cstdlib.

Note: Problems requiring the use of random numbers must properly seed the random number generator so that they generate different results each time they are run. (If you do not seed the random number generator, it gives you the same sequence of “random” numbers every time.)

Java
You will need a pseudorandom number generator to solve some problems. One such generator is provided by the `java.lang.Math.random()` static function, but here we discuss the more flexible `java.util.Random` class and, in particular, how it can be used to generate unsigned random integers.

First, create a generator object with
```
java.util.Random myGenerator = new java.util.Random();
```

In contrast to the C/C++ version, this is automatically seeded with the current time so that a different pseudorandom sequence is given on each run. If desired, a specific sequence can be requested by passing a `long` seed to the constructor.

Random integers are then generated as follows.
```
int myInt = myGenerator.nextInt(10);
// generates one of the 10 random integers between 0 and 9
```