Problems for Op 2008

By Eric Durant, PhD <durant@msoe.edu>
Friday 21 November 2008
Copyright © 2008 MSOE

1. Area of Triangle (10 Points)
The area, \( A \), of a triangle having sides of length \( a \), \( b \), and \( c \) is:

\[
s = \frac{1}{2}(a + b + c)
\]

\[
A = \sqrt{s(s-a)(s-b)(s-c)}
\]

Write a program that calculates and displays the area a triangle. The program will accept the lengths of the 3 sides as input from the user. The program may assume that the entries describe a valid triangle.

2. Quadratic Formula (10 Points)
Recall the quadratic formula...

\[
x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}
\]

which finds the solutions to the quadratic equation...

\[ax^2 + bx + c = 0\]

Write a program that accepts the coefficients \( a \), \( b \), and \( c \) as input. If these values would result in imaginary results (i.e., the radicand is negative), your program should state that and give no further output. Otherwise your program should display the 2 solutions to the equation. You do not need to handle the case of identical solutions (i.e., radicand of 0) as a special case.

3. Julian Day Number (10 Points)
The Julian Day Number is a unique, sequential integer for each date after 4800 BC. It has several useful properties, the key one being that the difference between day numbers is equal to the number of days between 2 dates. Write a program that accepts the month, day, and year as input from the user and displays the Julian Day Number. January is entered as 1, February as 2, etc. The full year must be used, which in most cases will be a 4-digit number. Years BC are entered as follows since there is no year 0: 1 BC (the year before 1 AD) is entered as 0, 2 BC as -1, 44 BC as -43, etc. You may assume that the user enters correct numbers for the month, date, and year. If we call the user-entered data month, day, and year, we can calculate the Julian Day Number according to the following formulas...

\[
a = \frac{14 - \text{month}}{12}
\]

\[y = \text{year} + 4800 - a\]

\[m = \text{month} + 12a - 3\]

\[JD = \text{day} + \frac{153m + 2}{5} + 365y + \frac{y}{4} - \frac{y}{100} + \frac{y}{400} - 32045\]
Note that these are integer divisions (remainders are discarded). Note also that y is not the same as year and m is not the same as month.

4. Depth of Field (20 Points)

In photography, the “depth of field” indicates how much of the subject of a photograph will appear to be in perfect focus. Sometimes a photographer wants “everything” to appear in focus, while other times, for emphasis, only a small portion of the image should appear in focus. When a lens is focused on something to be photographed, items a certain distance in front of (closer to the lens) and behind (farther from the lens) the subject will appear to be in focus, while items too close to the lens or too far from the lens will appear to be out of focus.

Using the formulas below, write a program that accepts information about the photo situation as input from the user (using the units specified for the formula below) and calculates and displays the distances from the lens between which the subject appears in focus. Also, calculate and display the difference between these distances, which is the range of distance that appears in focus.

We’ll assume our photographer’s vantage point and her subject are fixed. The only information about these positions that we require is $P$, the distance between the camera and the subject. The user will enter three more values...

1. One of these is $c$, formally known as the “circle of confusion diameter limit.” This number depends on a number of things like the distance at which the photograph will be viewed, how much it will be enlarged, etc., but a typical value is 0.03 mm (that is, only three hundredths of a millimeter).

2. The second number is $f$, the focal length of the lens, which is typically given in mm. A typical value for popular 35 mm digital SLR cameras is 50 mm. (“35 mm,” in this case, refers to the size of the film that the lens is designed for and does not directly enter into the calculation we are performing.)

3. The third number is $n$, the f-number, which is the ratio between the focal length and the iris diameter. The iris is an opening within the lens that can be made smaller to achieve deeper focus at the cost of letting less light in. A smaller iris results in a larger f-number.

The formulas are:

$$D_n = \frac{P f^2}{f^2 + 304.8 P n c}$$

$$D_f = \frac{P f^2}{f^2 - 304.8 P n c}$$

...where $P$ is in feet and $f$ and $c$ are in millimeters. $D_n$ is the “near focus distance,” the distance in front of the lens at which the scene first appears to be in focus, while $D_f$ is the “far focus distance,” the distance after which the scene appears to no longer be in focus.

Note: These are approximate equations. As a result, sometimes $D_f$ is negative for large values of $P$. That’s okay — your program will only be tested for “reasonable” values of inputs.

Remember that you also need to calculate the difference between these values, which is the “depth of field.” Both $D$ values are given in feet.
5. Prime Number Distances (20 Points)

Write a program that accepts an integer as input from the user and calculates the distance between the largest prime number less than or equal to the input and the smallest prime number greater than the input. You may use a library function to determine if a given number is prime, or you may write your own function to determine this. If the user enters a value less than 2 (e.g., 0 or 1; you do not need to be concerned with negative inputs or non-integer inputs), the program should print a warning message and prompt the user to enter another value until a valid value is entered.

For example, if the user enters 3, the output is 2 since the two adjacent primes are 3 and 5. As another example, if the user enters 290, the output is 10 since the two adjacent primes are 283 and 293.

6. Repetitions (20 Points)

Suppose, for example, we had sequenced a number of pieces of DNA and recorded the results in a source string: “CGATTACGCGACGAT”. Suppose also that we wanted to know how many times a particular target sequence appeared in the source string. For example, we might want to know how many times “CGA” appears in the string (3 in this case). It would be helpful to have a program that could examine a string and report the answer to this question for a given target.

Write a program that accepts two strings from the user and counts how many times the 2nd string (target) appears in the 1st (source). Consider another example: source “ATATATA” with target “ATA”. In this case, the correct answer is 3. Your program must count all instances of the target, even if they overlap with each other.

You are allowed to use any of the advanced string handling features of your language of choice, but this problem can be solved using only the basic string functions provided by Java and C++.

7. Bejeweled Calculator (40 Points)

The popular game Bejeweled removes colored jewels from a grid whenever the player arranges them so that at least 3 in a row horizontally or vertically have the same color. The user may only arrange jewels by swapping 2 adjacent jewels, and then only if it will result in at least 3 in a row having the same color, but these rules are not important for solving this problem. Write a function that determines whether a given board position has any stones that can be removed. Your function will accept a square, 2-D array of integers, from 1 to 7, where each number represents one of the 7 colors of stone. If it is helpful (given your language of choice), you may also have the function take the side length of the square as an input. You will output either true or false. true will indicate that there is at least one row or column of 3 stones of the same color that can be removed; false will indicate this is not the case. You will probably want to write a program to test your function, but only your function will be judged.

8. Degrees of Separation (40 Points)

Facebook wants to add a feature that lets you enter someone’s email address and, if they are in the system, tells you how many “degrees of separation” they have from you. If you are friends with someone, you have 1 degree of separation. You have 0 degrees of separation from yourself. You have 2 degrees of separation from all your friends’ friends who are not your direct friends or yourself. In the figure at right, for example, “0” and “2” have 2 degrees of separation, “5” and “5” have 0 degrees of separation, and “0” and “4” have 3 degrees of separation. “6” and “2” have infinity degrees of separation.
Write a program that takes a file from the Facebook server that has a list of all "friend pairs" in the system. To protect user identities, the file assigns each user a unique number from 0 to N-1, where N is the number of registered users. The file contains N on the first line, and a friend pair separated by a space on each additional line. After reading the file and performing the needed calculations, your program will prompt the user to enter 2 user IDs and then report the degrees of separation between them. For example, for the given diagram, the file might contain

7
0 1
2 1
2 4
3 2
5 6

Hints

- Keep track of the distances between people by using a matrix or 2-D array. Since the distance from a to b is the same as from b to a, you only need to use half of the matrix (perhaps the half where a ≤ b), but you can use the entire matrix.
- There are various ways to handle infinite degrees of separation, which is the case for every pair before considering any connections. For this program, use “9999” instead of infinity. When the result is “9999,” it will be understood that the users aren’t connected.
- Put “1”s in the matrix for each pair in the file. Now you have a matrix with 1s and 9999s but need to find where the 2s, 3s, etc. go.
- Iterate over the matrix as long as you find shorter connections. One correct algorithm is to consider each pair (a,b) and then consider all third parties c (you’ll probably need 3 or 4 loops to do this). If the degrees of separation (a,c) added to (b,c) are less than what you have for (a,b), then you’ve found a smaller number of degrees of separation for (a,b) and should update your matrix.

9. Modified Birthday Problem\(^1\) (40 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 entered 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, each containing 2 people, for a total of 20,000 people). 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.

\(^1\) The birthday problem is discussed at [http://mathworld.wolfram.com/BirthdayProblem.html](http://mathworld.wolfram.com/BirthdayProblem.html)
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 normally 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