Problems for Op 2015

By Eric Durant, PhD, MBA <durant@msoe.edu>
Friday 20 November 2015

- Note: Unless a problem specifies otherwise, you may assume that all user inputs are valid.
- Note: The precise wording of your outputs is not critical as long as they are completely clear.

1. Tip and split calculator (10 Points)

Write a calculator program that equally splits a restaurant bill between a given number of friends after adding a specified tip percentage. You can ignore rounding to the nearest cent since the purpose of the calculator is to give a very good approximation, not to be precisely accurate. But, you must display the amount in a standard dollars and cents format. Your calculator will prompt for the pre-tip total, the tip rate, and the number in the party; it will return the cost per person. For example, for a pre-tip total of $20.31, a tip of 18%, and a party size of 3, the exact answer is $7.9886, but you must display it as either $7.98 or $7.99 depending on how you chose to handle the rounding.

- You must give a meaningful prompt when asking for each piece of information.
- Don't forget to account for the percentage being an additional number of hundredths. For example, if the user enters 10.5 for the percentage, multiply the input dollar amount by 1.105.

2. Matching numbers (10 Points)

Write a program that allows the user to enter a series of positive integers; the program will report its result and end when 0 or a negative integer is entered. The program will report how many times the user entered the first positive integer. So, the program will report 0 if and only if the first integer entered is 0 or negative. For example, input of 5 8 5 2 5 0 gives output of 3, input of 9 -1 gives output of 1, and input of -7 gives output of 0.

3. Only one vowel (10 Points)

Write a program that allows the user to enter a word. You may assume the word is entered in lowercase and contains only letters. Indicate whether the word contains 1 unique vowel, even if that vowel is repeated. Vowels are defined as 'a', 'e', 'i', 'o', and 'u'. For example, "cat" and "goto" each contain 1 unique vowel, but "enlarge" does not and neither does "sh". You output will be either

- "The word contains 1 unique vowel." or
- "The word does not contain 1 unique vowel."

4. Third largest (20 Points)

Write a program that asks the user to enter a sequence of 10 floating point numbers. After the user enters the third through final numbers, display the third largest number from among all those entered so far. Duplicates count as unique places in the order. For example, if the first 5 numbers are -5.3, 3.7, -5.3, 4.8, 3.7, then the first 3 outputs are -5.3, -5.3, and 3.7.

5. Arithmetic-geometric mean (20 Points)

Write a program that calculates and displays the arithmetic-geometric mean of two positive, floating point numbers, a_1 and b_1. The arithmetic-geometric mean is the value to which the following series converges as *n* grows large…

$$a_{n+1} = \frac{a_n + b_n}{2}$$

$$b_{n+1} = \sqrt{a_n b_n}$$

You should stop and report a_{n+1} as the result as soon as it is within 0.01 of a_n after an iteration. Note that this is an "absolute error," not a "relative error." Do not run the iteration further as that computational effort will be considered wasted and will cause your solution to be rejected by the judges.

6. Juggler sequence (20 Points)

Calculate a juggler sequence, which is a series of positive integers, given a positive integer starting value. The next integer is always found by raising the integer to an exponent (resulting in a floating point intermediate result) and then rounding ***down***. The exponent is 0.5 (square root) if the integer is even, otherwise the exponent is 1.5. The series terminates immediately after 1 is reached and reported.

You ***must*** use an integer type such as `int` to hold the values of the series. Using a floating point type such as `double`, even if you round correctly, will give incorrect results since floating point types have precision greater than 1 for large values and thus cannot represent very large integers accurately.

There is one important error case that your program must handle. In some cases the number will reach or try to exceed the maximum integer value that can be represented (such as `Integer.MAX_VALUE` in Java or `INT_MAX` from `<limits.h>` in C++). When this happens, after reporting all numbers in the sequence up to the value that was too large, you should report "Series went too high." And end your program.

As a test case, you may want to try starting at 48443, which will result in the value going too high regardless of what intrinsic integer type you use.

7. Partition into three triangular numbers (40 Points)

Triangular numbers count the number of objects in a triangle of a given edge length. Any triangular number can be found using $T_n = \frac{n(n+1)}{2}$. Note that $T_0 = 0$ is a valid triangular number.

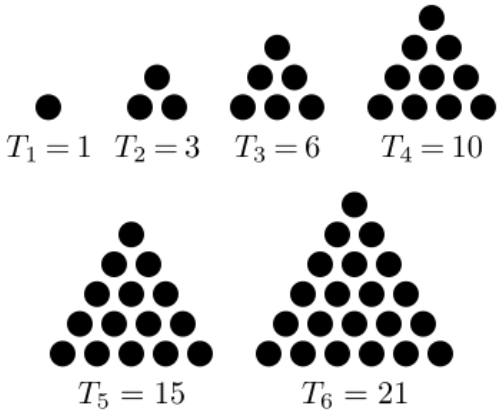

Figure Source: Wikipedia, CC BY-SA 3.0 Melchoir

It has been proven that any positive integer can be "partitioned" into 3 triangular numbers that yield the original integer when summed; in some cases there are many possible ways to partition a positive integer.

Your program will input a positive integer from the user and express it as a sum of 3 triangular numbers.

Hints:

- There are often many valid partitions and it is only necessary that you be able to find one for any given number.
- There may not exist a valid 3-way partition that includes the largest triangular number less than or equal to the given number, but this is a reasonable place to start your search.
- Triangular numbers may be repeated in the partition.
- 0 is a valid triangular number.

Examples:

- 1 can be partitioned as 1 0 0 or 0 1 0, or 0 0 1.
- 6 can be partitioned as 6 0 0 and 3 3 0, plus their permutations.
- 20 cannot be put into a partition including 15, which is the largest triangular number not larger than 20. But, 10 10 0 is a valid partition of 20.
- 30 can be partitioned numerous ways including 28 1 1, 21 6 3, 15 15 0, and 10 10 10.

8. Exchange rate arbitrage (40 Points)

Write a program that finds the best "triangular arbitrage" opportunity in currency conversions from and back to US dollars (USD). Read in a table of conversion factors for 8 major currencies. A file with the data below is provided to help you with testing, but the judges will test your program with other files.

	AUD	CAD	CNY	EUR	GBP	HKD	JPY	USD
AUD	1	0.952153	4.582447	0.634103	0.463361	5.576225	86.331528	0.720900
CAD	1.065123	1	4.940244	0.680883	0.495708	5.984590	91.643933	0.765404
CNY	0.218892	0.207801	1	0.140663	0.101551	1.209804	19.139644	0.157307
EUR	1.546365	1.456471	7.093427	1	0.750966	8.592669	135.297375	1.124500
GBP	2.125315	2.003891	9.778921	1.377917	1	11.708705	184.226658	1.532600
HKD	0.178693	0.166534	0.818408	0.116518	0.083673	1	15.725514	0.129032
JPY	0.011635	0.010769	0.053581	0.007419	0.005480	0.064916	1	0.008336
USD	1.387155	1.306500	6.357000	0.889284	0.652486	7.750000	119.968000	1

The rows represent the source currency and the columns represent the destination currency. For example, the value at the lower left shows that 1 USD yields 1.387155 AUD (Australian dollars).

A triangular arbitrage is done by converting USD to currency X, then currency X to currency Y, and finally currency Y back to USD. For example, if X is JPY and Y is EUR, we could do the following:

- Start with 1 USD
- Convert it to 119.968 JPY
- Convert each JPY to 0.007419 EUR, giving 0.890042592 EUR
- Convert each EUR to 1.1245 USD giving 1.000852894704 USD.

If $1,000,000 were converted this way the profit would be $852.89. High speed automated trading algorithms often take advantage of short-lived opportunities such as these.

Your program will

- Read in a tab-and-newline delimited currency conversion matrix that contains just the numbers in the same order as presented above for the 8 currencies (sample file on website at https://faculty-web.msoe.edu/durant/opcomp/data/). The currencies will not change and will always be presented in the order shown.
- Consider the profit for all combinations of triangular arbitrage starting from USD through 2 other currencies. You will not consider the cases where X=Y and/or X=USD and/or Y=USD and will therefore never use the diagonal elements, which all equal to 1 in the sample matrix.
- Report the maximum arbitrage opportunity as profit on $1,000,000. So, if the example above represents the maximum profit, report, "Converting $1,000,000 to JPY, then to EUR, and finally back to USD yields a profit of $852.89."
- The maximum profit can be negative. This can happen if the rates have fees embedded. Banks use this to claim that they are not charging currency conversion fees when in fact they adjust their rates to and from foreign currencies to include profit in each direction.

9. You scratch my back, I don't scratch yours (40 Points)

Write a program that simulates a population of birds and their ability to accumulate resources (food, health, etc.) that help them to be successful at reproducing. Unfortunately these birds are plagued by a parasite that attaches to their backs in a hard to reach spot. While a bird has no way on its own to rid itself of a parasite, the birds are resourceful and have learned that they can easily knock parasites off of each other. It does take a bit of work (depleting one's resources) to do so, however, and some of the birds would rather not bear that cost. There are 3 types of birds.

- Suckers: Suckers will help any fellow feathered friend who asks for their assistance in removing a parasite. If all the birds in a population are suckers, everyone is happy since they can always get their parasites removed by the first bird they ask.
- Cheats: Cheats never help other birds remove their parasites. A population that consisted of only cheats would suffer since no parasites would ever get removed, but in a population of mostly suckers with a few cheats, the cheats greatly benefit since they pretty much always get their parasites removed on the first try but never expend resources to remove a parasite themselves.
- Grudgers: The grudgers have caught on to the cheats. They start off by helping any bird who asks for help, but if they ask for help and are turned down, they never forget who cheated them. If that cheat ever asks the same grudger for help, the grudger turns him down. A population of just grudgers, or a population of suckers and grudgers without any cheats, always has everyone helping everyone else. In a population of just grudgers and cheats, the grudgers soon learn who the cheats are, resulting in the cheats never getting help but the grudgers getting help as long as they are lucky enough to ask another grudger.

In your computer model, the user will be asked to input several parameters that will control your simulation:

- Non-negative integers
    - s = number of suckers, from 0 to 20
    - c = number of cheats, from 0 to 20
    - g = number of grudgers, from 0 to 20
- Positive, floating point values
    - i = initial resources that each bird has
    - r = cost to a bird of helping another bird remove its parasite
    - p = cost to an infected bird of finding a random bird to ask to remove its parasite
- Positive integers
    - t = number of times a bird will ask to have its parasite removed before giving up and accepting an additional resource cost of 100×p.
    - N = number of rounds to simulate

Once the user enters all the parameters, you will first ensure that there are at least 2 birds in the population. If there are fewer than 2 birds, report the problem to the user and quit.

Then, you will conduct *N* rounds of simulation, keeping track of the resources that each bird has. You'll also need to keep track of which birds each grudger is grudging against.

Note that resources for some or all birds might become negative. That is okay for this simulation.

In each round, you will iterate through all the birds (suckers, then cheats, then grudgers). Each bird will get infected by a parasite and then try up to *t* times to ask a random other bird to remove the parasite. Be careful that you don't have a bird try to remove its own parasite since that is not allowed! The random bird is selected "with replacement" meaning that the same other bird may be asked multiple times for help by the same parasitized bird in a single round. Each time the bird asks for help it incurs a cost *p* and if it finds a bird to help it that bird incurs a cost *r*. If the bird still has its parasite after *t* attempts it gives up, lives with the parasite, and incurs an additional cost of 100×*p*.

After all the rounds are completed, provide the following report to the user, substituting the results from your simulation.

- There were 10 suckers and they ended up with average resources of 13.43. The resource values were: <list of the 10 numbers goes here>
- There were 5 cheats and they ended up with average resources of 4.87. The resource values were: <list of the 5 numbers goes here>
- There were 20 grudgers and they ended up with average resources of 13.79. The resource values were: <list of the 20 numbers goes here>

You should test a variety of the situations described above. For example, you might test with no suckers, 15 cheats and 15 grudgers, costs of 2 to help and 1 to ask, up to 10 tries, and 50 rounds. With these settings you will find that the grudgers have a huge advantage.

Generating Random Integers

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

double myDouble = myGenerator.nextDouble();
// generates a random value between 0 and 1
```

C++

You will need 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, starting it 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 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 **`int`**. The function **`rand()`** returns integer values between 0 and **`RAND_MAX`** where **`RAND_MAX`** is a constant defined in **`cstdlib`**. So, **`(double)rand() / RAND_MAX`** will return a random **`double`** between 0 and 1.

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.