MTH 4300
Spring 2025
E. Fink

Homework 2

---

1) nfactors.cpp

Write a C++ program which asks the user to input a positive integer `n`. The program should print out the least positive integer which has exactly `n` positive integer factors.

For example, if the user entered 8, the program would output 24, because 24 has exactly 8 positive integer factors (1, 2, 3, 4, 6, 8, 12, and 24) and no positive integer less than 24 has exactly 8 factors.

Your code must use a `while` loop, at least one function that you've written (other than `main()`), and the mod operator. (I can't think of a reasonable way to do this problem which doesn't satisfy those criteria!)

Please comment the function(s) you write in a style like that recommended in class: describe WHAT the function returns, being clear how it relates to the function parameters, and state what you assume of the arguments when you use the function. If you have trouble describing what the function does, it *might* be an indication that you should rethink the function you are trying to write. The most useful functions often have a clear and easily-described purpose.

Also, please use good style generally: indent roughly like you would in Python, use descriptive-but-not-overly-long variable names, and try to be consistent in your placement of open and close braces (admittedly, I have not been a good model of consistency in this matter).

Specifications: your program must

- ask for and accept a positive integer N from the user.
- print out the least positive integer with exactly N factors.
- use a `while` loop, at least one function that you've written (other than `main()`), and the mod operator.
- at least attempt reasonable style – including commenting what your function does. (I will not penalize as long as a clear attempt has been made!)

---

2) greentao.cpp

An *arithmetic progression* is a sequence of numbers where the difference between any two consecutive terms is the same. For example, $5, 8, 11, 14, 17$ is an arithmetic progression, since $8 - 5 = 11 - 8 = 14 - 11 = 17 - 14 = 3$ (the value 3 is called the *common difference*).

One of the biggest breakthroughs in 21st century mathematics is the Green-Tao theorem, which states that there exist arithmetic progressions of arbitrary length where all the terms are prime numbers. For example, $7, 19, 31, 43$ is an arithmetic progression of length 4 where all terms are prime. The theorem says that if you look hard enough, you can definitely find an all-prime arithmetic sequence of length 10, of length 100, of length 100000, of any length at all.

I guess it wouldn't be fair to ask you to verify a Fields-medal winning Theorem. So instead, a related problem: write a program that finds the *longest arithmetic sequence composed of primes, where the first two terms are less than 1000.* Please print out the length of the sequence, along with the terms in the sequence. If there is a tie – i.e., two such sequences with the same length – you only have to print out one sequence.

Some hints: I think you would want to steal my `is_prime()` function from class. (You can take it just as I wrote it or improve on it, your choice.) Furthermore, you should write at least one other function – after all, using triply-nested loops is rarely a good look, and I don't see how to answer this problem without either a triply-nested loop or some sort of helper function. I can provide recommendations for functions that might be helpful, but I encourage you to try to imagine them for yourself.

In this problem, please still try to employ the stylistic advice given for the last problem, regarding function commenting, indenting, variable names, and bracing.

Specifications: your program must

- not ask for any input from the user.
- print out (one of) the longest arithmetic progression(s) whose first two terms are less than 1000 consisting of only prime integers.
- write at least one function, other than my `is_prime()` function.
- maintain good style, as recommended in the previous problem.

3) fire.cpp

A forest fire has broken out! Will it be contained in 7 days?

In my file `fire.cpp`, I have put some starter code, which includes an array called `forest`. This array contains 12 `char` entries. The first and last entries represent the edge of the forest; the ten entries in the middle represent 10 sites in the forest, all lined up in a row, consisting of 10 trees, one of which is on fire.

Each entry in `forest` represents the state of each site, which will either be `'x'` representing empty ground (which includes the forest edge); `'t'` representing a standing tree; or `'F'` representing a tree on fire.

You will simulate 7 afternoons and evenings. Each **afternoon**, the fire may spread, by the following rule:

> for each site (you may ignore the edges), if that site currently contains a tree in the morning, and at least one of its neighbors directly to the left or right is on fire in the morning, then the site will catch fire this afternoon with probability 0.4.

And each **evening**, the fire may die down a bit, by the following rule:

> for each site (you may ignore the edges), if that site currently is on fire in the afternoon, then the fire will burn out at that site this evening with probability 0.2, in which case the site will become empty ground.

To deal with probabilities, simply use `dist(generator)` to generate random `double`s between 0 and 1. `dist(generator) <= 0.4` will be true with probability 0.4, and `dist(generator) <= 0.2` will be true with probability 0.2.

Estimate the probability that there is still at least one site on fire at the end of 7 days. You can do this by having your program run a single simulation, and print out the final state of the forest; and then run your program 50 times, and count the number of times that there is at least one site still on fire. If that sounds silly to you, you could instead just program a simulation loop; this is encouraged. Put your estimate of the probability as a comment in your code.

A warning about an easy-to-miss bug in the afternoon updates: consider a configuration of four consecutive sites which looks like

F, t, t, t

in the morning. At the end of the afternoon, the second site may be on fire, but the third one definitely shouldn't be.

However, if you're not careful, there will be a bug, as follows. Your code for the afternoon would probably would visit the sites left-to-right. The first site is already on fire, so nothing happens in the afternoon. Then you move to the second site and since it has a neighbor on fire, it is very possible that the second site sets on fire, leading to the state

F, F, t, t

Now, your loop moves on to the third site. Since a neighbor of the third site is NOW on fire, if you haven't programmed carefully, it is possible for the third site to change to F as well, when it shouldn't.

To avoid this bug, you can create a separate "morning" array at the beginning of each day, which captures the state of the forest in the morning, and use the morning array to update the `forest` array.

Specifications: your program must

- not ask for any input from the user, and contain my starter code.
- update the 10 middle entries of the `forest` array using the rules described above, for 7 "days" (each with an afternoon and evening update), using the random number generator to produce random values where needed.
- at the end, print out the values in the `forest` array (not necessary if you estimate the probability using a simulation loop).
- put your estimate of the probability that at least one site is still on fire after 7 days in a comment. [I'm unlikely to grade on the accuracy of your estimate.]

(This question is inspired by `http://nifty.stanford.edu/2007/shiflet-fire/`.)