Programming Project 03

This assignment is worth 30 points (3% of the course grade) and must be **completed and turned in before 11:59 on Monday, February 6th .**

Assignment Overview
This assignment will give you more experience on the use of loops and conditionals, and introduce the use of functions.

Background
There are all sorts of special numbers, much like perfect numbers that we saw in Python. Take a look sometime at http://mathworld.wolfram.com/topics/SpecialNumbers.html for a big list. One such class of numbers is called *Achilles Numbers.* Great name! An Achilles has two special features based on its ***prime factors*** using some unusual named properties of numbers:

- the number should be ***powerful***.
- the number should ***not be a perfect power***.

Lets look at all three of those properties

Prime Factors
The prime factors of any integer should be familiar. You find all the prime integers that divide exactly (without remainder) into an integer. We exclude 1 (which is a factor of every integer) and the number itself (which, again, is a prime factor of every particular number).

- for example, the prime factors of 30 are 2,3,5 since 2*3*5=30. Note that 6, 15 and 10 are also factors but they are **not prime** factors.
- take a look at https://en.wikipedia.org/wiki/Table_of_prime_factors for a table of the prime factors of many numbers.

Powerful Number
A powerful number is an integer with the following property. For every prime factor in the number, the square of the prime factor must divide exactly (without remainder) into the original number.

- 30 **is not** a powerful number. Its prime factors are 2,3,5. 2^2 does not divide exactly into 30, 3^2 does, 5^2 does not. All of the factors squared must divide exactly into the number for it to be powerful.
- 72 **is** a powerful number. Its prime factors are 2,3 ($2^3 * 3^2 = 72$). 2^2 divides exactly into 72 (18 * 4) as does 3^2 (9 * 8).

https://en.wikipedia.org/wiki/Powerful_number gives a list of powerful numbers.

Perfect Power
A perfect power number is an integer that can be expressed as a single expression of the form m^k. That is, there is some **integer** `m` which, raised to the **integer** power `k`, is the **integer** `n` in question.

Project Description / Specification
Warning
First, a warning. In this and in all future projects we will provide *exactly* our function specifications: the function name, its return type, its arguments and each argument's type. The functions will be tested using the main we provide for you. If you do not follow the function specifications, these independent tests of your functions will fail. Do not change the function declarations!

Provided Main
Since we provide the main, there is a file `skeleton.cpp` that you should use to start your program. It contains the provided main, you simply add your functions to that file.
Warning!!! You have to use the provided main in your code. If you modify the provided main in any way you will receive a 0 for the project.

Functions
function: `is_prime`: return is `bool`. Argument is a single `long n`. If `n` is prime it returns `true`, otherwise it returns `false`.

function: `is_powerful`: return is `bool`. Argument is a single `long n`. If `n` is powerful it returns `true`, otherwise it returns `false`. Utilizes `is_prime` (or should).

function: `is_perfect_power`: return is `bool`. Argument is a single `long n`. If `n` is a perfect power it returns `true`, otherwise it returns `false`.

function: `is_achilles`: return is `bool`. Argument is a single `long n`. If `n` is an Achilles number it returns `true`, otherwise it returns `false`. Utilizes `is_powerful` and is `perfect_power` (or should).

Input and Output
We provide 4 test cases (`input1.txt, input2.txt, input3.txt, input4.txt`) that, respectively, test each of the functions. We also provide an output (`output1.txt, output2.txt, output3.txt, output4.txt`) for each of the 4 inputs so you may test your functions individually.

Deliverables
`proj03.cpp` -- your source code solution including the provided main (*remember to include your section, the date, project numberand comments in this file*).

1) Be sure to use "proj03.cpp" for the file name (or handin might not accept it!)
2) Save a copy of your file in your CS account disk space (H drive on CSE computers). This is the only way we can check that you completed the project on time in case you have a problem with handin.
3) Electronically submit a copy of the file.

General Hints:
1. You can write as many functions are you like over and above the ones I have specified.
    a. Make sure you write the requested functions exactly as specified. They will be tested individually according to that specification.
2. The functions all return Booleans, which is not very informative. Feel free to place lots of output statements in your functions so you can see what is going on.
    a. Just remember to remove the output statements before you turn in the code!!!
3. The main will take in test cases to test each function. You already have the main, so develop the functions one at a time and run the appropriate test case to see that the function works!
    a. Don't write everything all at once, write one function at a time, test it and make sure it works.

Specific Hints
If we were more knowledgeable about algorithms we could discuss how to be efficient about these checks, but for now we would just like them to work. Here are some pretty specific suggestions, but feel free to work it out for yourself

`is_prime`:

- You can check every number from 2 up to the n-1 and see if any of those numbers divides without remainder. If so, then it isn't prime. Otherwise it is.
    - you can be more efficient. What value do you really need to check up to (less than n-1)?
    - `break` can save you some computational time here.

`is_powerful`:

- First, no prime number is ever powerful.
- Check every number from 2 to n-1
    - if it divides without remainder into n, then it is a factor,
        - for all of those factors that are prime, then
            - if the square of that factor also divides without remainder into n, it is a powerful number
        - otherwise not powerful

`is_perfect_power`:
There are a couple ways to approach tis:

- check all combinations of base to some power that makes sense
    - Outer loop goes from 2 to some limit. This is the base
    - Inner loop. raise base to power. powers start at 2. check 2^2, then 2^3, then 2^4 until the value exceeds the value of n. inner loop ends, check the next base: (3^2, 3^3, etc.)
- check all the integer power roots
    - check powers from 2 to some limit
    - find the various roots (square root, cube root, etc.) by taking the `pow(n, 1.0/power)`.
    - turn the root into an integer (`round` in `cmath` is helpful)
    - see if the now integer root raised to the power being checked is equal to n.

`is_achilles`:

- if n is powerful but not a perfect power, then it is an Achilles number