# Project One: Integers!

## Out: May 20, 2021; Due: May 27, 2021

## Background

Haruhi is talking with Koizumi about numbers. They don't know whether 775249 is a prime number. Then, they start to talk about the properties of integers. However, Haruhi says she cannot solve those problems individually. And she decides to let you help her.

## Motivation

This project will give you experience in using basic C++ constructs including I/O, arithmetic operators, branch, and loop.

## Introduction

Integers have many properties. For example, they can be odd, even, prime, or composite. In this project, we will test the following four properties of a given integer:

1. **Armstrong number**: a positive integer *n* is called an Armstrong number if for all positive integers *b* that are smaller than *n*, $b^n \ mod \ n = b.$ For example, 3 is Armstrong number since $1^3 \ mod \ 3 = 1, 2^3 \ mod \ 3 = 2.$

2. **Cyclone Jet**: an integer is called a cyclone jet if the number reads the same both ways. For example, 121 and 4334 are cyclone jet.

3. **Black Premium Car number**: an integer *n* is called a black premium car number if $x = \frac{\sqrt{24n+1}+1}{6}$ is an integer. The number 6 is not a black premium car number because $x = \frac{\sqrt{24\times6+1}+1}{6} = 2.17$ is not an integer.

4. **Auspicious number**: an integer is called an auspicious number if the sum of its proper divisors exceeds the integer. Note that a proper divisor of an integer *n* is a positive divisor of *n*, excluding *n* itself. Thus, 12 is an auspicious number, because the sum of its proper divisors is 1+2+3+4+6 = 16 > 12. However, 28 is not, since 1+2+4+7+14 = 28.

## Programming Assignment

You will implement a program that tests whether a given integer has a specific property.

## Input/Output

Your program should first prompt:

`"Please enter the integer and the test number: "`

**You must use exactly this prompts. Don't forget the trailing single space! We recommend you use the function provided in starter file to avoid potential error.**

Then your program will take two integers as inputs, separated by a white space. The first one is an integer to be tested and the second one is an integer between 1 and 4, denoting one of the above properties to be tested for. The first integer should be a **positive integer** and be **no larger than** 10 million. The second input should be in the range between 1 and 4, inclusively. **If either input entered is outside its range, your program should prompt the above statement and take the inputs again. (You should not prompt anything other than the above statement.)** You can assume that the user always enters integral values, not any other erroneous inputs (i.e., you can always read the value into a variable of `int` type). Assume the entered values are within the range from -20,000,000 to 20,000,000.

Your output will be either 0 or 1, where 1 indicates that the test succeeds and 0 indicates that the test fails.

Thus, the input and output will look like:

```
Please enter the integer and the test number: 3 1
1
```

Below is a situation where the first input attempt fails.

```
Please enter the integer and the test number: -1 1
Please enter the integer and the test number: 3 1
1
```

Note the prompt of the statement for the second time because the first value you input at the first time is illegal (negative).

## Implementation Requirements

You should put **all** of the functions you write in a single file, called `p1.cpp`. You may only include <iostream>, <cmath>, <string>, and <cstdlib>. No other system header files may be included, and you may not make any call to any function in any other library.

## Compiling and Testing

To compile, type the following Linux command:

```
g++ -Wall -o p1 p1.cpp
```

You should test your program extensively.

## Submitting and Due Date

You only need to submit your source code file p1.cpp (**name it exactly like this!**). The source code file should be submitted via the online judgment system. The due date is 23:59, May 27, 2021.

## Grading

Your program will be graded along three criteria:

1. Functional Correctness

2. Implementation Constraints

3. General Style

An example of Functional Correctness is whether or not you produce the correct output. Implementation Constraints checks whether you stick to the implementation requirements. General Style speaks to the cleanliness and readability of your code. We don't need you to follow any

particular style, as long as your style is consistent and clear. Some typical style requirements include: 1) appropriate use of indenting and white space, 2) program appropriately split into subroutines, 3) variable and function names that reflect their use, and 4) informative comments at the head of each function.