Project One: Integers!

Out: May 27, 2017; Due: June 4, 2017

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. **Fibonacci number**: an integer is a *Fibonacci number* if it is one of the numbers in the *Fibonacci sequence* $a_0, a_1, \ldots, a_n, \ldots$, where $a_0 = 0, a_1 = 1$, and for any $n \geq 2$, $a_n = a_{n-1} + a_{n-2}$.
2. **Sum of consecutive squares**: an integer is a *sum of consecutive squares* if the integer is equal to $m^2 + (m+1)^2 + (m+2)^2 + \cdots + n^2$ for two integers $0 \leq m \leq n$.
3. **Repeated number**: an integer is a *repeated number* if it is positive and it looks like another positive integer repeating for at least twice. For example, 111 and 1212 are repeated numbers, but 123 and 1212123 are not. Note that by convention, the leftmost digit of a positive integer is not a 0.
4. **Divisor-sum number**: An integer is a *divisor-sum number* if it is positive and the sum of its proper divisors is equal to the integer. Note that a proper divisor of a positive integer *n* is a positive divisor of *n*, excluding *n* itself. For example, 28 is a divisor-sum number, since the sum of its proper divisors is 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 test choice: ”`

Then, your program will read the test choice. After that, it should prompt

`“Please enter the number for test: ”`

Then, your program will read the number for test.

You must use these prompts exactly. Do not forget the trailing single space!

The test choice is an integer between 1 and 4, indicating one of the above four properties to be tested for. Therefore, the first input should be in the range between 1 and 4, inclusively.

The second input is the number that your program should test for the given test choice. It should be a **non-negative integer** and be **no larger than** 10 million.

If either input entered is outside its range, your program should prompt the request for that input and take the input again. This should be repeated until the input is valid. (You should not prompt anything other than the prompt.) 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 `Pass` or `Fail`, depending on whether the number passes the test or not.

An example of the input and output is:

```
Please enter the test choice: 4

Please enter the number for test: 28

Pass
```

An example of wrong input attempts is:

```
Please enter the test choice: -1

Please enter the test choice: 4
Please enter the number for test: 28

Pass
```

Note the program prompts "`Please enter the test choice: `" twice, because the first value you input at the first time is illegal (negative).

Another example of wrong input attempts is:

```
Please enter the test choice: 4

Please enter the number for test: -1

Please enter the number for test: -2

Please enter the number for test: 28

Pass
```

Implementation Requirements

You should put **all** of the functions you write in a single file. 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

Write a `Makefile`. Put all your compiling commands in the `Makefile`. **The output program should be named `p1`.**

You should test your program extensively.

Submitting and Due Date

You should submit your single source code file and the `Makefile`. These two files should be submitted via the online judgment system. Please follow TA's announcement for submission details. The due date is 11:59 pm on June 4th, 2017.

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 do not 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.