The following are solutions to problems found on the Project Euler website. The problem descriptions are given in the code itself. Although this sheet may contain a restatement of the problems, it is mostly intended for a more thorough and mathematical explanation of the solutions as opposed to bare code. Not all problems may appear. For benchmarking purposes, here is the relevant system data for the machine that all programs were tested on. The source code can be found at `https : //github.com/patl1/project_euler`.

CPU info:

| | | |
|---|---|---|
| vendor id | : | GenuineIntel |
| model name | : | Intel(R) Core(TM)2 Duo CPU P8600 @ 2.40GHz |
| cpu MHz | : | 800.000 |
| cache size | : | 3072 KB |
| cpu cores | : | 2 |

Other info:

| | | |
|---|---|---|
| Operating System | : | Fedora 18 (Spherical Cow) |
| Kernel Version | : | 3.9.6-200.fc18.x86_64 |
| Memory | : | 4GB DDR3 SDRAM, 1066 MHz |
| GCC Version | : | gcc (GCC) 4.7.2 20121109 (Red Hat 4.7.2-8) |

## Problem 1

**Multiples of 3 and 5**: If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23. Find the sum of all the multiples of 3 or 5 below 1000.

Firstly, the classic summation formula for a series of numbers is $\sum\limits_{i=1}^{n} i = \frac{n(n+1)}{2}$. In general, this can be extended to $\sum\limits_{i=a}^{b} i = \frac{b(a+b)}{2}$ where $b$ is the maximum number and $a$ is the minimum number (proof omitted). Since here the summation is every third or fifth number, we have that the number of terms to sum is not $b$ but $\frac{b}{a}$. For ease of reading, we define $n = \lfloor \frac{b}{a} \rfloor$. Then we have $\sum\limits_{\substack{i=a \\ i=i+m}}^{b} i = \frac{n(a+b)}{2}$ where $m$ is the multiple. With a little bit of algebraic manipulation, we arrive at $\sum\limits_{\substack{i=a \\ i=i+m}}^{b} i = \frac{na(1+\frac{b}{a})}{2} = \frac{na(1+n)}{2}$. While it seems trivial, replacing $(a+b)$ with $a(1+n)$ is necessary to insure that the summation has the correct number of integers and does not accidentally include a number that is too high because of rounding errors. So now we have $\sum\limits_{\substack{i=a \\ i=i+m}}^{b} i = \frac{na(1+n)}{2}$. To find the summation of all multiples of three and five, we can use this formula for $m = 3$ and $m = 5$. However, this double-counts numbers where $i$ is a multiple of both 3 and 5 (a multiple of 15). So we subtract out all multiples of fifteen to go from double-counting them to single-counting. The final mathematics looks like this:

$$\sum_{\substack{i=1 \\ i=i+3}}^{999} i + \sum_{\substack{i=1 \\ i=i+5}}^{999} i = \frac{\lfloor\frac{999}{3}\rfloor(3)(1+\lfloor\frac{999}{3}\rfloor)}{2} + \frac{\lfloor\frac{999}{5}\rfloor(5)(1+\lfloor\frac{999}{5}\rfloor)}{2} - \frac{\lfloor\frac{999}{15}\rfloor(15)(1+\lfloor\frac{999}{15}\rfloor)}{2}$$

Assuming one's button-pushing skills are correct, this expression should evaluate to 233,168.

## Problem 2

The iterative solution is relatively basic, and needs no explanation.

The slightly more mathematical solution is based on the ratios of the Fibonacci numbers. As the Fibonacci sequence approaches infinity, the ratio of the one number to the next in sequence approaches the golden ratio, $\phi = \frac{1+\sqrt{5}}{2} \approx 1.61803398875$. We use the fact that $\frac{F_{n+1}}{F_n} \approx \phi$ for all integers $n$. Every third number in the sequence is even, since the sum of an even and an odd must be odd, and the sum of two odds must be even. Then we can see that

$$\begin{aligned} \frac{F_{n+3}}{F_n} &= \frac{F_{n+3}}{F_{n+2}}\frac{F_{n+2}}{F_n} \\ &= \frac{F_{n+3}}{F_{n+2}}\frac{F_{n+2}}{F_{n+1}}\frac{F_{n+1}}{F_n} \end{aligned}$$

and since for all integers $n$, $\frac{F_{n+1}}{F_n} \approx \phi$,

$$\begin{aligned} \frac{F_{n+3}}{F_n} &= \left(\frac{F_{n+3}}{F_{n+2}}\right)\left(\frac{F_{n+2}}{F_{n+1}}\right)\left(\frac{F_{n+1}}{F_n}\right) \\ &= (\phi)(\phi)(\phi) \\ &= \phi^3 \end{aligned}$$

Therefore, the ratio of each even number to the next even number in the sequence is roughly $\phi^3$.

The performance methods between the two methods is of interest. The iterative method definitely executes more instructions as it goes through all elements of the Fibonacci sequence, whereas the golden ratio method does not. However, the golden ratio method uses floating point numbers, which are inexact and take more time to compute on average. Each calculation must also be rounded to the appropriate even number so that the calculation is correct, another step which takes time to execute.

The two methods were timed to see which would run faster. The results are summarized below. For benchmarking info and machine specs, please refer back to the beginning of the paper. Timings were taken using the Linux `time` command.

Time results for `fib_iter`:

```
real  0m0.003s
user  0m0.002s
sys   0m0.002s
```

Time results for `fib_with_phi`:

```
real  0m0.003s
user  0m0.002s
sys   0m0.001s
```

Both implementations were very quick. The elapsed real time and user space CPU time were identical, and the golden ratio method was slightly quicker in system CPU time. This difference is only 0.001 seconds, which is not enough to say that this method is definitely faster—the difference could be explained by some very small factor like system load factor or page faults, although these are unlikely. Logically, no conclusions can be drawn about which program is "better". Perhaps if the upper limit were to go up to a number much greater than 4,000,000 there would be a more noticeable difference, if one exists.

## Problem 3