A Perfect Problem

Some numbers are better than others

Perfect Numbers

- Find all of the perfect numbers
  - Equal to the sum of its factors, eg. 6 = 1+2+3

```
consider each candidate {
  consider each possible factor of the candidate {
    if it is actually a factor {
      add to the running total
    }
  }
  if the total of factors equals the candidate {
    bingo!
  }
}
```

Perfect C++

```
/**
 * Find perfect numbers
 */

#include <iostream>
#include <cstdlib>

using namespace std;

/**
 * Prints perfect numbers up to a specified limit.
 */
int main(int argc, char* argv[])
{
    const int limit = 1000;
    cout << "Perfect numbers up to " << limit << "..." << endl;
    for (int candidate = 2; candidate < limit; candidate++) {
        int total = 1;
        for (int factor = 2; factor < candidate; factor++) {
            if (candidate % factor == 0) {
                total += factor;
            }
        }
        if (total == candidate) {
            cout << candidate << endl;
        }
    }
    cout << "That's all!" << endl;
}
```

cstdlib contains standard C library functions (like **atoi**)

```
Perfect numbers up to 1000...
6
28
496
That's all
```

What about up to 10000 or 100000?

More Perfect

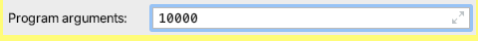
```
/**
 * Find perfect numbers
 */

#include <iostream>
#include <cstdlib>

using namespace std;

/**
 * Prints perfect numbers up to a specified limit.
 */
int main(int argc, char* argv[])
{
    const int limit = argc > 1 ? atoi(argv[1]) : 1000;
    cout << "Perfect numbers up to " << limit << "..." << endl;
    for (int candidate = 2; candidate < limit; candidate++) {
        int total = 1;
        for (int factor = 2; factor < candidate; factor++) {
            if (candidate % factor == 0) {
                total += factor;
            }
        }
        if (total == candidate) {
            cout << candidate << endl;
        }
    }
    cout << "That's all!" << endl;
}
```

C++ equivalent of Java's **String[] args**
argc is the number of arguments
argv are the values (the first of which is the program name itself)

Control the search limit—make **limit** a parameter

Run from the command line/terminal:
$ ./perfect 10000

or, edit the build configuration and run within CLion:



Even More Perfect

```
...
for (int factor = 2; /* when to stop? */; factor++) {
    if (candidate % factor == 0) {
        total += /* when to add? */;
    }
}
...
```

- Make it run faster
  - Stop sooner: fewer factors to try
  - If you find a factor, you have also found a co-factor
- How much faster?

Perfect cofactors

```
/**
 * Find perfect numbers
 */

#include <iostream>
#include <cstdlib>

using namespace std;

/**
 * Prints perfect numbers up to a specified limit.
 */
int main(int argc, char* argv[])
{
    const int limit = argc > 1 ? atoi(argv[1]) : 1000;
    cout << "Perfect numbers up to " << limit << "..." << endl;
    for (int candidate = 2; candidate < limit; candidate++) {
        int total = 1;
        for (int factor = 2; factor*factor < candidate; factor++) {
            if (candidate % factor == 0) {
                total += factor + candidate/factor;
            }
        }
        if (total == candidate) {
            cout << candidate << endl;
        }
    }
    cout << "That's all!" << endl;
}
```

cofactor

How much faster?

- Profile the code
  - Run original version for various limits
  - Use a smarter algorithm and re-run
  - Use system clock to time

limit	Original (s)	Cofactor (s)
1000	0.002	0.001
10000	0.165	0.003
100000	15.7	0.076
1000000	1578	2.28
10000000	!!	74.0

As good as it gets?

```
for (int candidate = /* generate the sequence to test */) {
    ...
}
```

- Still more speed?
  - Number theory: reduce the number of candidates to test
    Perfect numbers follow certain patterns
  - All are “triangular”: 1, 3(=1+2), 6(=1+2+3), 10(=1+2+3+4), ...
  - All are sums of odd cubes: 1^3 , ($1^3 + 3^3$) , ($1^3 + 3^3 + 5^3$), ...
  - All obey Euclid's formula: $2^{n-1}(2^n - 1)$: 1 , 6 , 28 , 120, …

Perfect Euclid

```
Perfect numbers up to
9223372036854775807...
6
28
496
8128
33550336
8589869056
137438691328
2305843008139952128
That's all
```

Euclid: only need to check series $2^{n-1}(2^n - 1)$
6, 28, 120, 496, 2016, 8128, 32640, ...

```
#include <iostrream>
#include <climits>
#include <cmath>

long sumOfFactors(long n);

int main(int argc, char* argv[])
{
    const long limit = argc > 1 ? atoi(argv[1]) : LONG_MAX;
    cout << "Perfect numbers up to " << limit << "..." << endl;
    long candidate;
    for (int n = 2; (candidate = pow(2,n-1)*(pow(2,n)-1)) < limit; n++) {
        if (sumOfFactors(candidate) == candidate) {
            cout << candidate << endl;
        }
    }
    cout << "That's all!";
}

long sumOfFactors(int n)
{
    long total = 1;
    for (long factor = 2; factor*factor < n; factor++) {
        if (n % factor == 0) {
            total += factor + n/factor;
        }
    }
    return total;
}
```

limit	Original	Cofactor	Euclid
1000	0.002	0.001	
10000	0.165	0.003	
100000	15.7	0.076	
1000000	1578	2.28	
10000000	!!	74.0	0.000
100000000000000	!!	!!	0.117