10th Marathon of Parallel Programming

SBAC-PAD – 2015

October 19th, 2015.

Warmup Rules for Local Contest

For all problems, read carefully the input and output session. For all problems, a sequential implementation is given, and it is against the output of those implementations that the output of your programs will be compared to decide if your implementation is correct. You can modify the program in any way you see fit, except when the problem description states otherwise. You must upload a compressed file with your source code, the *Makefile* and an execution script. The script should have the name of the problem. You can submit as many solutions to a problem as you want. Only the last submission will be considered. The *Makefile* must have the rule *all*, which will be used to compile your source code before submit. The execution script runs your solution the way you design it – it will be inspected not to corrupt the target machine.

All *Local Teams* have access to the target machine during the marathon. Your execution may have concurrent process from other teams. Only the judges have access to a non-concurrent environment.

The execution time of your program will be measured running it with *time* program and taking the real CPU time given. Each program will be executed at least three times with the same input and the mean time will be taken into account. The sequential program given will be measured the same way. You will earn points in each problem, corresponding to the division of the sequential time by the time of your program (*speedup*). The team with the highest points at the end of the marathon will be declared the winner.

This problem set contains 7 problems; pages are numbered from 1 to 21.

Problem A

Harmonic progression sum

The simplest harmonic progression is

$$1/2, 1/3, 1/4, 1/5, \cdots$$

Let $S_n = \sum_{i=1}^{n}(1/i)$, compute this sum to arbitrary precision after the decimal point.

Input

The input contains only one test case. The first line contains two values: the first is the number of digits *D* and the second is the value of *N*. Consider ($1 \leq D \leq 10^5$) and ($1 \leq N \leq 10^8$).

The input must be read from the standard input.

Output

The output contains only one line printing the value of the sum with exact *D* precision.

The output must be written to the standard output.

Example

Input	Output for the input
`12 7`	`2.592857142857`

```cpp
#include <iostream>
#include <sstream>

using namespace std;

void sum(char* output, const long unsigned int d,
const long unsigned int n) {
  long unsigned int digits[d + 11];
  for (long unsigned int digit = 0; digit < d + 11;
++digit) {
      digits[digit] = 0;
  }
  for (long unsigned int i = 1; i <= n; ++i) {
      long unsigned int remainder = 1;
      for (long unsigned int digit = 0; digit < d +
11 && remainder; ++digit) {
            long unsigned int div = remainder / i;
            long unsigned int mod = remainder % i;
            digits[digit] += div;
            remainder = mod * 10;
      }
  }
  for (long unsigned int i = d + 11 - 1; i > 0; --
i) {
      digits[i - 1] += digits[i] / 10;
      digits[i] %= 10;
  }
  if (digits[d + 1] >= 5) {
      ++digits[d];
  }
  for (long unsigned int i = d; i > 0; --i) {
      digits[i - 1] += digits[i] / 10;
      digits[i] %= 10;
  }
  stringstream stringstreamA;
  stringstreamA << digits[0] << ".";
  for (long unsigned int i = 1; i <= d; ++i) {
      stringstreamA << digits[i];
  }
  stringstreamA << '\0';
  string stringA = stringstreamA.str();
  stringA.copy(output, stringA.size());
}

int main() {

  long unsigned int d, n;

  cin >> d >> n;

  char output[d + 10]; // extra precision due to
possible error

  sum(output, d, n);

  cout << output << endl;

  return 0;
}
```