INSTRUCTIONS: This assignment contains three problems. Please submit your source code as follows:
– for Problem 1, write your code in a file named a5harm.cc
– for Problem 2, write your code in a file named a5perfect.cc
– for Problem 3, write your code in a file named a5prime.cc
Use the instructions from the course lab page to prepare and submit your files using Moodle.
A) Please use your template.cc file for each of the specified files and then fill in the required information. If you do not have a template.cc file please copy the one from the library. The command to copy is: "`cp $L/samples/template.cc ~/assn`". Please see Arie if you have any questions regarding this.
B) Comment your variable declarations by specifying their role.
C) Comment your functions by specifying the role of the function, of the parameters, and of the return value if applicable.

PROBLEM 1: The Harmonic number H_n appears frequently in the analysis of algorithms. It is defined by

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n} = \sum_{i=1}^{n} \frac{1}{i}.$$

Write first a C++ function with the prototype `double harmonic(double n)` which returns H_n.
Write then a program that prompts the user to enter a value for n as a variable of type `double`. The program should then print the values for H_n, $\ln n$ (the natural logarithm of n) and the difference $H_n - \ln n$.
Examples:

```
Please enter a value for n 1000
H_n = 7.48547
ln(n) = 6.90776
H_n - ln(n) = 0.577716

Please enter a value for n 67000
H_n = 11.6897
ln(n) = 11.1124
H_n - ln(n) = 0.577223
```

Observations & hints:
– Note that the type of the parameter for function `harmonic` is `double` which allows us to approximate H_n for values of n larger than what `int` can represent (and thus experience the consequences of exponential running time behaviour).
– To calculate the natural logarithm $\ln n$, use the library function `double std::log(double n)`. You must include the proper header file with the command `#include <cmath>`.
– Calculations involving type `double` are approximate. Rounding errors will accumulate. For example, the type double can accurately represent rational numbers with about at most 17 decimal digits (`http://en.wikipedia.org/wiki/Double-precision_floating-point_format`).
– To instruct `std::cout` to print more than 6 decimals for a value of type `double`, use the I/O manipulator `setprecision(d)` (include `<iomanip>`):
`std::cout << std::setprecision(17);`

PROBLEM 2: An integer p is called a divisor of another integer n if p divides n. For example, 3 is a divisor of 6. All the divisors of 6 are 1, 2, 3, and 6. A divisor p of n is called *proper* if $p \neq n$. The proper divisors of 6 are 1, 2, and 3. The only proper divisor of 5 is 1, and the proper divisors of 24 are 1, 2, 3, 4, 6, 8, and 12.
An integer n is called deficient if the sum of its proper divisors is less than n. Example: 5 is deficient

because 1 is less than 5. An integer n is called perfect if the sum of its proper divisors is equal to n. Example: 6 is perfect because $1 + 2 + 3 = 6$. Finally, an integer n is called abundant if the sum of its proper divisors is greater than n. Example: 24 is abundant because $1 + 2 + 3 + 4 + 6 + 8 + 12 = 36 > 24$. Write a computer program prompts the user to enter a single integer and determines whether the integer is deficient, perfect, or abundant as in the examples.
Examples (corresponding to 6 runs of the program):

```
Please enter an integer: 4
4 is deficient
Please enter an integer: 6
6 is perfect
Please enter an integer: 5
5 is deficient
Please enter an integer: 13
13 is deficient
Please enter an integer: 24
24 is abundant
Please enter an integer: 82
82 is deficient
```

PROBLEM 3: An integer p is called prime if its only divisors are 1 and p. Write a computer program that reads one integer from input and prints whether that integer is prime or not, as in the examples.
Examples— (corresponding to two successive runs of the program):

```
Please enter an integer: 141
141 is not prime
Please enter an integer: 7919
7919 is prime
```