02393 C++ Programming Exercises

Assignment 2

To be handed in via CodeJudge — `https://dtu.codejudge.net/02393-e21/assignments`

1 Gaussian Sum

Write a program that reads a positive integer n and prints the result of the sum $1 + 2 + \ldots + n$. For example, for $n = 100$ the result is 5050.

2 Prime Factorization

Write a program that reads a positive integer and prints its prime factorisation. For instance, the factorisation of 60 is $2 * 2 * 3 * 5$.

Hints.

- In C++, the modulus function % gives the remainder of integer division, i.e., x is divisible by y if and only if $x\%y == 0$.
- Given the number n to factorize, iterate through all the numbers $i = 2, 3, 4, 5, \ldots$ and check whether n is divisible by i. If so, print out "$i *$ " and continue to check the factorization of n/i. Stop when n cannot be further factorised.

When testing your solution on CodeJudge, please ensure that *(1)* the factors are printed in ascending order, *(2)* between two factors there are always a space, an asterisk ($*$), and another space, and *(3)* after the last factor there is a newline (`endl`).

3 Approximating π

Compute an approximation of π using the Leibniz formula:

$$\frac{\pi}{4} = \sum_{i=0}^{\infty} \frac{(-1)^i}{2i+1} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \ldots$$

To this purpose, write a function `double pi(int n)` that computes the first n terms of the infinite summation above, and returns the sum of such n terms multiplied by 4. Example: for $n = 1$ we get 4 (a bad approximation of π); by increasing n, the approximation of π gets better.

Hints. To compute an exponentiation x^y, you may use the function `pow(x,y)`, which is available by including `math.h`

Note that "i" (in the Leibniz summation) and "n" are two different things:

- for $n = 1$ you should compute the term 1 of the Leibniz summation
- for $n = 2$ you should compute the terms 1 and $-\frac{1}{3}$ of the Leibniz summation, and sum them
- for $n = 3$ you should compute all the previous terms and $\frac{1}{5}$, and sum them
- ...

4 Remembering numbers

Write a program that reads a positive integer value between 0 and 1000, and:

- if the given value is between 1 and 1000 (included), then the program prints how many times that value was already given before; then the program repeats, by reading another value;
- if the given value is 0, the program terminates without printing anything.

For example, an execution of the program may look like:

`10`	*(user input)*
`0`	*(program output: 10 was given 0 times before)*
`42`	*(user input)*
`0`	*(program output: 42 was given 0 times before)*
`10`	*(user input)*
`1`	*(program output: 10 was given 1 time before)*
`10`	*(user input)*
`2`	*(program output: 10 was given 2 times before)*
`701`	*(user input)*
`0`	*(program output: 701 was given 0 times before)*
`0`	*(user input; the program terminates, without printing anything)*

Hint: an array could be used to keep track of how many times each number between 1 and 1000 has been given...