Lab 3. Computing large numbers.

As you know, the numeric type `int` in C++ is only an approximation of the mathematical numbers due to its finite representation as a sequence of bits. The goal of this lab is to observe `Integer Overflow` and try resolve the caused problems.

In this lab, we are going to compute the number of combinations $C(n,k)$. If you are already familiar with this notion and with the factorial formula for computing it, you may skip the introduction. Otherwise, please read the introduction.

Introduction: Computing the number of combination.

The number of ways to **select** k **items out of** n **available** is called the **number of combinations** and is denoted

$$\mathbf{C(n,k)} \quad \text{or} \quad \binom{\mathbf{n}}{\mathbf{k}}.$$

For instance, if you have to choose two from the following list of activities: $\{Sleep, Studying, Games\}$, then there are exactly three ways to do that:

$$\{Sleep, Studying\},\ \{Sleep, Games\},\ \text{and}\ \{Studying, Games\}.$$

Because you were choosing 2 out of 3 things, this is described by $C(3,2)$, and

$$C(3,2) = 3.$$

How to compute $\mathbf{C(n,k)}$?

$C(n,k)$ can be computed using the following formula (for $0 \le k \le n$):

$$\boxed{C(n,k) = \frac{n!}{k! \cdot (n-k)!},}$$

where the notation $m!$, the factorial of m, stands for the product of all numbers from 1 to m:

$$\begin{aligned} 1! &= 1 \\ 2! &= 1 \cdot 2 \\ 3! &= 1 \cdot 2 \cdot 3 \\ \dots \\ m! &= 1 \cdot 2 \cdot 3 \cdot \ldots \cdot m \end{aligned}$$

And by convention, we also say that:

$$0! = 1$$

So, the full formula is computed as follows, for example:

$$C(10,3) = \frac{10!}{3! \cdot (10-3)!} = \frac{10!}{3! \cdot 7!} = \frac{(1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 \cdot 7 \cdot 8 \cdot 9 \cdot 10)}{(1 \cdot 2 \cdot 3)\ \cdot\ (1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 \cdot 7)} = 120.$$

Of course, computing these multiplications is not too difficult when we can use loops. How many loops would you need, what do you think?

Useful properties of $\mathbf{C(n,k)}$

Property P1:

$$\boxed{C(n,\ n-1) = n, \qquad C(n,\ 1) = n}$$

What does it mean? It says that, for instance, you can select 4 out of 5 items in $C(5,4) = 5$ ways. For instance, if the given items are $\{a,b,c,d,e\}$, then possible selections are:

$$\{a,b,c,d\},\ \{a,b,c,e\},\ \{a,b,d,e\},\ \{a,c,d,e\},\ \{b,c,d,e\}.$$

Similarly, to select 1 item out of 5, there are also $C(5,1) = 5$ ways:

$$\{a\},\ \{b\},\ \{c\},\ \{d\},\ \{e\}.$$

It is not a coincidence that both numbers $C(5,4)$ and $C(5,1)$ are the same, because the process of selecting 4 out of 5, $C(5,4)$, is essentially the process of "*unselecting*" or deleting 1 out of 5 items, which can be done in $C(5,1)$ ways.

And it generally holds that $C(n,\ n-1) = C(n,\ 1) = n$.

Property P2:

$$\boxed{C(n,\ n-k) = C(n,\ k)}$$

This property holds, because both left-hand side and right-hand side are equal to $\frac{n!}{k!(n-k)!}$. And intuitively, the task of selecting k items is equivalent to the task of "unselecting" the other $(n-k)$ items, as was discussed above for the Property P1.

Implementing $\mathbf{C(n,k)}$ in C++?

Using the property P1, we can immediately tell that $C(1000000, 999999) = 1000000$. However, can we use computer to confirm that? Yeah, it seems so, we can calculate $C(1000000, 999999)$ directly using the provided factorial formula:

$$= \frac{10000000!}{999999! \cdot 1!} = \text{... What would we get?}$$

To do that we can first compute all three factorials, 1000000!, 999999!, and 1! and then find their ratio. Right?

It turns out, computing it correctly in C++ is not a trivial task. If we use integer types (such as `int` or `long int`) for storing factorials, these types can hold only limited range of numbers. So if the actual factorials are larger than those limits, our direct computation will fail.

When you try it, you will see that direct computation of $C(n,k)$ will fail even for relatively small arguments n and k. Floating-point types also have limited precision so cannot be reliable either.

Therefore, we will need more intelligent algorithms for computing $C(n,k)$, and this is the topic of this lab.

Programming assignment.

Task A. Direct approach

Write a program "**lab3a.cpp**" that

- asks the user to input n,
- asks the user to input k,
- computes $C(n,k)$ using the formula and prints it on the screen.

$C(n,k)$ can be computed directly by finding the 3 factorials: $n!$, $k!$, and $(n-k)!$:

$$C(n,k) = \frac{n!}{k!\cdot(n-k)!} = \frac{(1\cdot 2\cdot 3\cdot \ldots \cdot n)}{\big(1\cdot 2\cdot 3\cdot \ldots \cdot k\big)\big(1\cdot 2\cdot 3\cdot \ldots \cdot (n-k)\big)}$$

You probably want to do this by writing three loops. Use type `int` to represent numbers.

Testing: According to the property **P1**, $C(2,1)=2$, $C(3,1)=3$, $C(4,1)=4$, and so on ... ideally. However, our program is not perfect. What is the smallest value of n for which our program returns an incorrect result when computing $C(n,1)$? Put it in the comments. Also in the comments, explain why the results are incorrect (you don't need to figure out the exact relevant bit patterns).

Task B. Optimization I: Skipping some factors.

Write a new program "**lab3b.cpp**" that has the same user interface, but uses a better algorithm.

Observe that in the original formula

$$C(10,3) = \frac{10!}{3!\cdot(10-3)!} = \frac{10!}{3!\cdot 7!} = \frac{(\cancel{1\cdot 2\cdot 3\cdot 4\cdot 5\cdot 6\cdot 7}\cdot 8\cdot 9\cdot 10)}{(1\cdot 2\cdot 3)\,\cdot\,(\cancel{1\cdot 2\cdot 3\cdot 4\cdot 5\cdot 6\cdot 7})} = \frac{8\cdot 9\cdot 10}{1\cdot 2\cdot 3}$$

we have the same numbers in the numerator and the denominator, and they can be canceled out right away, and not computed at all! Can we save some bits this way and avoid integer overflow?

$$C(n,k) = \frac{(\cancel{1\cdot 2\cdot 3\cdot \ldots \cdot (n-k)}\cdot (n-k+1)\cdot \ldots \cdot n)}{\big(1\cdot 2\cdot 3\cdot \ldots \cdot k\big)\big(\cancel{1\cdot 2\cdot 3\cdot \ldots \cdot (n-k)}\big)} = \frac{(n-k+1)\cdot \ldots \cdot n}{1\cdot 2\cdot 3\cdot \ldots \cdot k}$$

Testing: Repeat the tests for the new program. It is most definitely better, but is it perfect? Can it reliably compute $C(n,n-1)$? Write your observations in the comments.

Also, can you estimate the efficiency of this algorithm? Specifically, how many multiplications does your program do when the input is (n,k)?

Task C. Optimization II: Symmetry of C(n,k).

Write a new program "**lab3c.cpp**" improving the previous one using the property **P2**:

$$C(n,k) = C(n,n-k).$$

(Hint: The question about the efficiency of the program in the task B can help you. Remember that you want to minimize the number of multiplications your program performs. So, given that you have to compute $C(n,k)$, sometimes it can be cheaper to switch to computing $C(n,n-k)$, if it requires fewer multiplications.)

(Hint 2: Let's take a specific example of $C(11,2)$ and $C(11,9)$, they are numerically equal, however, for your program B they will require significantly different number of multiplications. Take a piece of paper and count how many multiplications does it need for each of them. One case is easy to compute, the other is hard. We want our program be smart to always switch to the easier case.)

Testing: Repeat the same tests as in task A and B. Write your observations in the comments.

Task D. Get a fancier formula? Reordered factors.

The formula for $C(n,k)$ can be also expressed as the following product:

$$C(n,k) = \Big(1+\frac{n-k}{1}\Big)\Big(1+\frac{n-k}{2}\Big)\Big(1+\frac{n-k}{3}\Big)\ldots\Big(1+\frac{n-k}{k}\Big)$$

(no magic here, it is a rearrangement of the factors after the common terms in the numerator and denominator were canceled, like we did in task B).

Write a program "**lab3d.cpp**" to implement this new formula. It should be able to handle much larger values of n and k now. Because the factors in the formula are non-integers, you should use the floating-point type `double` for the product and be careful with the division, maybe you will need to use some type-casting too.

Test your code and comment.

Task E. `long int` instead of `int`.

Copy your task A program in a new file "**lab3e.cpp**" and change it to use type `long int` instead of `int` when computing factorials. Do the results improve and if yes, by how much?

Test the code and comment.

How to submit your programs (the same requirements)

Each program should be submitted through Blackboard (Course Materials > Lab).

You can submit all your programs at the end of the lab session (or any time before the deadline). Basically, submit when you are sure that it will be your final version.

Write separate programs for each part of the assignment. For example, you can call the files `lab1a.cpp`, `lab1b.cpp`, and `lab1c.cpp` for the parts A, B, and C. Attach only source code files, not the compiled executables.

Each program should start with a comment that contains your name and a short program description, for example:

```
/*
Author: <your name>
Course: {135,136}
Instructor: <their name>
Assignment: <title, e.g., "Lab 2">

Here, briefly, at least in one or a few sentences, describe what the
program does.
*/
```

You can submit incomplete programs, but their “incomplete” status must be clearly mentioned in the comment to the program. In this case, also briefly describe what is implemented, and what is not.

Grading

All 135 and 136 programs this semester will be graded on:

- *Correctness*: Does your program work?
- *Testing*: Have you generated sufficient and good test data to give reasonable confidence that your program works?
- *Structure*: Have you structured your code to follow proper software engineering guidelines? This includes readability and maintainability. Note that
- *Documentation*: How well documented is your code? Good documentation does not repeat the code in English, but explains the point of each code block, highlighting any design decisions and/or tricky implementation details.