(Hi-tech summary of low-tech lecture.)

k-subsets: $\text{ksub}(S,k) = \{H \subseteq S \mid |H| = k\}$.

How to compute it? One idea: "filter" the power set:

```
for (H ∈ P(S)) {
    if (|H| == k)
        // add H to answer
}
```

$2^{|S|}$ steps! ☹

but computing $\mathcal{P}(S)$ is expensive. Seems unnecessary.

Indeed, we can do better by computing it "directly".
Same idea as $\mathcal{P}(S)$ though: fix $x \in S$ & partition the answer ksub(S,k) as follows:

$$\text{ksub}(S,k) = \text{ksub}(S \setminus \{x\}, k) \cup \text{ksub}(S \setminus \{x\}, k-1) \oplus x$$

(k-subsets w/o x)

(union of sets)

The $C \oplus x$ notation means "add x to each set in C" where C is a set of sets!

E.g., if $C = \{\{1,2\}, \{2,3\}\}$ & $x = 0$,
then $C \oplus x = \{\{1,2,0\}, \{2,3,0\}\}$

Note: $C \oplus x$ should be easy to construct in C++:

```
for (iterator i = C.begin(); i != C.end(); i++) {
    set<int> T = *i;
    T.insert(x);
```

```
        (C0x).insert(T);
    }
```

We saw something very similar for the power set.

What about the base case? I think we'll need 2 of them:

if k == 0, return {{}}.

if k > |S|, return {}.

Try to program this. It's a very good exercise. Kind of challenging though!

One more example from combinatorics: permutations.

Want to compute all "rearrangements" of a list/vector.

E.g., if V = [0, 1, 2], then

perms(V) = [[0,1,2], [1,0,2],
[0,2,1], [2,0,1],
[1,2,0], [2,1,0]]

Here's the breakdown: give each element a "turn" being last, & then add it to each permutation of the remaining elements. See above. Indeed,

= perms([0,1])

= perms([0,2])

= perms([1,2])

This is a little tricky to program, but you can do it! Use vectors. I'll give you an outline. Again, this is challenging, but worth doing.