Pairs

Here are the programmer pairs for today’s lab. Drivers are on the left.

- Muhammad Hamza and Amanda Hinchman-Dominguez
- Marcel Champagne and Madeleine Goldman
- Elizabeth Zak and Lindsey Byrne
- Devin Dooley and Linda Oyolu
- Corey Simmonds and Danielle Williams
- Michael Owusu and Bazil Mupisiri
- Jong Hoon Bae and Kevin Connors
- Thu Nguyen and Ella Nicolson
- Adam Hudson and Matt Jasaitis
- David Chang and Julia Fay
- Tyler Williams and Dennis Chan
- Adam Wesely and Kathryn Yetter
- Samee Zahid and Anita DeWitt

Jacob Ekstrand will replace anyone who is absent.

Constructing the Permutations of a Class

A permutation of a class $C$ is a sequence with no repetitions containing every member of $C$ (and no other values). For instance, the sequence $\langle 3, 2, 5, 1, 0, 4 \rangle$ is a permutation of the class $\{0, 1, 2, 3, 4, 5\}$.

As we have seen, the factorial function counts the permutations of a class of a specified cardinality. The factorial is defined by the recursion equations

\[
0! = 1, \\
(n + 1)! = (n + 1) \cdot n!
\]
These equations hint at a possible algorithm for constructing the permutations of a class \( C \): If \( C \) is empty, its only permutation is the empty sequence. Otherwise, choose the first element \( c \) of a permutation in every possible way (taking \( c \) to be each member of \( C \) in turn); then construct every permutation of \( C - \{c\} \) and extend every such permutation by adding \( c \) at the front.

**Exercise 0:** Define and test a Scheme procedure `class-permutations` that implements this algorithm, taking any class as argument and returning the class of all of its permutations.

**Constructing the Combinations of a Class**

A combination of \( k \) members from a class \( C \) of cardinality \( n \) is simply a \( k \)-member subclass of \( C \). We have seen that the binomial coefficient \( \binom{n}{k} \) counts such combinations.

**Exercise 1:** Define and test a Scheme procedure `class-combinations` that takes a class \( C \) and a natural number \( k \) as arguments and returns the class of all the \( k \)-member combinations of members of \( C \). The reading “Permutations and Combinations” contains a prose description of the algorithm to use.

**Bag Permutations**

Now let’s consider starting with a bag rather than a class. Since sequences, like bags, can contain duplicates, it makes sense to construct permutations from the members of bags. For instance, \( \langle b, a, b, b, c \rangle \) is a permutation of the bag \([a, b, b, b, c]\).

However, when a bag contains duplicate members, the number of possible permutations is less than the number of permutations of a class of the same size, because rearranging duplicates does not change the identity of a permutation – it’s still the same values in the same order, so it’s still the same sequence. In the extreme case, a bag consisting of nothing but duplicates, such as \([k, k, k, k]\), has only one permutation (namely \( \langle k, k, k, k \rangle \)), not 4!.

If we adapt the algorithm for constructing class permutations that we implemented above in a naive way, our implementation could spend a lot of
time constructing duplicate permutations, only to discard all of those duplicates when we assemble the permutations into a class. It would be better not to construct the duplicate permutations to begin with. Fortunately, in adapting the \texttt{class-permutations} algorithm to bags, we can add a step to ensure that duplicate permutations are never constructed in the first place, no matter how many duplicate values there are in the bag.

\textbf{Exercise 2:} Figure out what that change is and implement it in order to define a \texttt{bag-permutations} procedure.

\section*{Counting Bag Permutations}

The “Permutations and Combinations” reading proposed a counting rule for bag permutations: The factorial of the sum of the multiplicities of the values in the bag, divided by the product of the factorials of those multiplicities.

\textbf{Exercise 3:} Write and test a definition of a \texttt{bag-permutation-count} procedure that takes any bag as its argument and returns the number of permutations of the members of that bag.

\textbf{Exercise 4:} Use your rule to count the permutations of a bag containing 12 \texttt{a}s, 8 \texttt{b}s, 3 \texttt{c}s, and 1 \texttt{d}.