In this lab, we'll look at some ways of using one-dimensional arrays (the C analogue of vectors in Scheme) in programming.

**Pairs**

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

- Faizaan Ali and Tyler Williams
- Eli Salm and Anna Blindermann
- Saung Thuya and Jong Hoon Bae
- Jae Hong Shin and Lex Martin
- Josh Lavin and Lilly Webster
- Dennis Chan and Tanner Tufto
- Gemma Nash and Yuyin Sun
- Zachary Susag and Erhaan Ahmad
- Mattori Birnbaum and Ella Nicolson
- Ying Zhang and Sophie Gaschott
- Colin Greenman and Cory McCartan

Sanjay Sudhir will replace anyone who is absent.

**Exercises**

The functions `rForward`, `rBackward`, `rTurnLeft`, and `rTurnRight` have the same signature: Each one takes two arguments, both of type `double`, the first representing the speed the movement operation and the second representing its duration. We can use this congruence of signatures to store an entire robot dance in three arrays.

The first will be an array of characters. Each element of this array will designate a motion function by one of the capital letters in its name: 'F' for `rForward`, 'B' for `rBackward`, 'L' for `rTurnLeft`, 'R' for `rTurnRight`. So, for example,

```c
char motions[] = { 'F', 'L', 'F', 'R', 'F', 'L', 'F', 'R' };
```

describes a serpentine dance pattern alternating forwards motions with turns in alternate directions.

The other two arrays will be arrays of doubles, of length equal to the length of the character array. One will give the speeds of the successive motions, the other their durations. So, for example,

```c
double speeds[] = { 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0 };
double durations[] = { 1.5, 0.2, 1.5, 0.2, 1.5, 0.2, 1.5, 4.0 };
```

 describes motions that all take place at top speed but durations that are alternately long and short until the last motion, which goes on for much longer than any of the others.

Taken together, these three arrays define a dance consisting of four forward motions, each a second and a half long, separated by short turns (a fifth of a second each), alternately left and right, and culminating in a four-second clockwise circular motion.

Exercise 000: Write, compile, and run a program that includes the given array declarations and initializers and executes the dance they describe. Use a `for`-statement to walk through the arrays “in parallel,” taking one element from each array on each iteration of the body of the `for`-loop. Use a `switch`-statement to choose which motion function to
invoke on the basis of the character in the \texttt{motions} array. (For example, on the first iteration of the \texttt{for}-loop, the switch statement should evaluate \texttt{motions[0]} to figure out which of the motion functions to call first, and should give that function \texttt{speeds[0]} and \texttt{durations[0]} as its arguments.)

Exercise 001: Without changing the values in the arrays, rewrite the \texttt{for}-loop so that all the movements are carried out at half speed, but for twice as long.

Exercise 010: Again without changing the values in the arrays, rewrite the \texttt{for}-loop so that the movements are carried out in the opposite order (beginning with the long circular motion). This involves traversing the array from right to left rather than from left to right.

Exercise 011: Again without changing the values in the arrays, rewrite the \texttt{for}-loop so that the movements are all the opposite direction (forwards motions become backwards motions, left turn become right turns, and so on).

Exercise 100: Replace the initializers in the array declarations with array constants of your choice to produce a different dance.

Exercise 101: Add a fourth array, \texttt{pitches}, with the same number of elements as the others, and modify the \texttt{for}-loop so that the robot generates tones of the specified pitches and durations as it moves. You’ll need to make all of the calls to the movement functions “non-blocking.” Achieve this without modifying the \texttt{durations} array.

Exercise 110: Modify the program so that, after completing the dance, the robot performs the same music again (i.e., generates the same sequence of pitches, with the same durations), but this time without the dance.

Exercise 111: Modify the program so that, after completing the dance, the robot performs the same music again, but one octave higher and twice as fast. (As usual, achieve this without modifying the \texttt{pitches} or \texttt{durations} arrays.)