Lab session 3: Again with the dot products

- **In all of what follows, print the results of each calculation to `cout` so you can easily check that you're getting reasonable numbers.**
- **Write a (slightly silly) templated function which takes two arguments and returns their product, and calculate a product of two floats, of two doubles, and of two ints using this method.**
- **Satisfied with our mastery of templates, we now return to the dot product. Use Thrust `host_vectors` and `device_vectors` to make two integer arrays in device memory. As before, take the length of the vectors as a command-line argument, and fill them with randomly generated integers.**
- **Make a third `device_vector`, and use `transform` to fill its elements with the products of corresponding elements of the other two vectors. (In other words, `results[i] = vec1[i]*vec2[i]`).**
- **Now use `reduce` to get back the sum of the elements of the third vector, which is the dot product of the first two vectors.**
- **Use `inner_product` to take the dot product in a single kernel launch.**
- **Make a struct having an `operator` method which takes as its argument a `tuple` of two integers, and returns their product.**

- **Using a `zip_iterator` to combine iterators over the two `device_vectors` into a single tuple iterator, repeat the `transform_reduce`, with your new struct as the unary operator, to calculate the dot product.**
- **Write a struct which has:**
  - **An integer member variable `wrt` and**
  - **An `operator` method that takes a single integer `x`, and returns `x % wrt`.**
- **Using the above struct as your unary operator, write a `transform_reduce` call which will calculate the sum of the entries mod 2 of one of your integer vectors. This is equivalent to counting the number of odd entries.**
- **Repeat the above exercise in a different way: Instead of having the `wrt` variable be a member of the struct, let its argument be a tuple of two integers, and let it return the modulus of the first with respect to the second. Then write yet another `transform_reduce` call, in which you use a `zip_iterator` to combine your vector of ints with a `constant_iterator`.**