

# Lab 10

## Preview

*30 March 2017*

Today you'll start development on a project that provides a (small) library of classes to a potential user. Specifically, it will be a group of classes that store elements without duplication—a set.

### Sets

What is a set? Its fundamental properties are that it

- contains elements,
- does not count or distinguish duplicates, and
- does not guarantee anything about their order.

That means that it can't, for instance, retrieve an element at a particular index, because indices imply order and sets don't (promise to) preserve order. Think about it, and in your notebook, write down the key methods that a `Set` class will have to have. (If you're a little stuck, you might refer back to the `UnsortedType` definition in Chapter 3, which is not identical but is quite similar.) There are three or four really important ones, plus a few that would be more optional. Make sure to mark which ones would be `const`.

Once you're pretty confident about your list, write a file `Set.h` that encodes this information in the form of valid C++ method headers. We would like to make our `Sets` able to hold any type of element; at this point, it's time to start using templates for that. To make that happen, you just need to precede the class header with

```
template <class Thing>
```

and then use `Thing` as the name of the type the `Set` would hold, whenever you add a value or search for a value or anything like that. (Feel free to use a different name than `Thing`.)

Because our `Set` class is meant to define an interface, we want to mark its methods as “pure virtual”: we’ve mentioned this in class, but implementing it is just a matter of marking the method `virtual` and setting the body to zero. That is, if you had written a method

```
int getSomeValue() const;
```

you would mark it pure virtual by writing

```
virtual int getSomeValue() const = 0;
```

Write a simple test file called `test_VectorSet.u` that, for now, just `#includes` your `Set.h` file and has an empty test suite. Compile that file to confirm that your header has no errors.

### Test cases

Now that we have a public interface, we can start planning our test cases. In your notebook (*not* yet in the `.u` file), describe a few useful examples (which will eventually become the test fixture). Then, write some sequences of method calls, using those examples, that collectively verify that a `Set` would correctly contain its elements, and does not count or distinguish duplicates.