Lab 9: Generics and uC

Learning objectives:

- Understand how generics with implicit constraints work in a language like C++
- Understand how generics with explicit constraints work in a language like Java
- Understand how to write programs in uC

Use the following commands to download and unpack the distribution code:

```
$ wget https://eecs390.github.io/lab/lab09/starter-files.tar.gz
$ tar xzf starter-files.tar.gz
```

1. *C++ function templates.* Implement the `max_iterator()` function template. When called on a non-empty container that supports the iterator interface, and where the elements are comparable with the `>` operator, `max_iterator()` should return an iterator pointing to the maximum element in the container.

   Use `std::begin()` and `std::end()` (defined in the `<iterator>` standard header) to obtain begin and end iterators -- they work on class-type containers as well as built-in arrays. You may need to make use of the `auto` keyword in your solution.

   ```
   int main() {
     std::vector<int> nums = {0, 3, 2, 7, 9, 1};
     assert(*max_iterator(nums) == 9);

     std::list<std::string> strs = {"hello", "world", "zello"};
     assert(*max_iterator(strs) == "zello");

     double dnums[] = {3.14, 1.618, 2.718, 1.414};
     assert(*max_iterator(dnums) - 3.14 <= 0.0001);
   }
   ```

   Write your solution in `max_iterator.hpp`.

   To compile and run the tests:

   ```
   $ g++ --std=c++20 max_iterator_test.cpp -o max_iterator_test.exe
   $ ./max_iterator_test.exe
   ```

2. *Java generics.*

   a) The file `Array.java` contains the definition of a multidimensional-array class that can hold elements of type `String`. Read through the provided code. Then make whatever modifications are necessary to make `Array` a generic class, so that it can hold elements of arbitrary class type.

   The `Array` class is built by storing elements in a single-dimensional built-in Java array. Variadic methods are used to allow the `Array` to have any valid rank (i.e. dimensionality), and the `indexOf()` method translates a multidimensional index to a single-dimensional location in the underlying Java array.

   The `main()` method of `ArrayTest` provides some tests, and the expected output from running the test is in `ArrayTest.correct`.

   We have provided a repl.it project for this question so that you don't have to install Java locally. Click the "Fork Repl" button in the top right corner to make a copy of it for yourself. (You will need to sign in via Google/GitHub or create an account).

   When you're ready to test your implementation, compile and run the test driver (the compiler may provide some warnings, but should still compile your code):

```
$ javac ArrayTest.java
$ java ArrayTest
```

b) Once you have a working generic `Array` class, implement the `indexOfMax1D()` method of the `Util` class in `Util.java`. This method takes in a generic `Array`, which must be rank one and have at least one element. It returns the index of the maximum element in the array.

To be able to compare elements to each other, the element type must implement the appropriate `Comparable` interface. You will need to modify the header of `indexOfMax1D()` to enforce that this is the case.

The `main()` method of the `UtilTest` class contains some tests. Compile and run as follows:

```
$ javac UtilTest.java
$ java UtilTest
```

3. *Sorting in uC.* In the following exercise, you will implement a merge sort in uC. Write your answers in the given `mergesort.uc` file. You will not be able to run your implementation because you have not yet done project 5. Instead, compare your implementation to the staff solution when it is released.

a) Implement the `slice()` function, which takes in an array of integers, a start index, and an end index and returns a new array containing the elements from `[start, end)`. You may assume that `start` and `end` are within `[0, array.length)` and that `start < end`.

```
int[] arr = new int[]{1, 2, 3, 4, 5};
slice(arr, 0, 3); // returns [1, 2, 3]
slice(arr, 2, 4); // returns [3, 4]
slice(arr, 0, 0); // returns []
```

b) Implement the `merge()` function, which takes two arrays of integers and returns a new array that is the merge of the two arrays' elements in ascending order.

```
int[] a = new int[]{3, 7, 12};
int[] b = new int[]{1, 4, 6, 7};
merge(a, b); // returns [1, 3, 4, 6, 7, 7, 12]
```

c) Implement the `mergesort()` function, which takes an array of integers and returns a new array that contains the elements of the original array sorted in ascending order.

```
int[] arr = new int[]{3, 7, 6, 9, 2, 5, 0};
mergesort(arr); // returns [0, 2, 3, 5, 6, 7, 9]
```