CS 311 Fall 2017
Assignment 2

Assignment 1 is due at **5 pm** ~~Thursday, September 21~~ **Saturday, September 23**. It is worth 25 points.

Procedures

This assignment is to be done individually.

Turn in answers to the exercises below on the UA Blackboard Learn site, under Assignment 2 for this class.

- Your answers should consist of the source code for Exercise A (file `ksarray.h`). This should be **attached** to your homework submission.
- Send only the above! I do not want project files or executables.
- I may not look at your homework submission immediately. If you have questions, e-mail me.

Exercises (25 pts total)

Exercise A — Kinda Smart Array Class

Purpose

In this exercise, you will write a simple class that manages an array. It will not be quite as “smart” as it could be (`std::vector` is much smarter), but it will be much better than a built-in C++ array; for example, it knows its size, and has copy operations.

Key to this assignment is proper management of a resource in a class. Be sure you have no resource leaks, and that ownership and related issues are properly documented.

And as before, make your code high quality. In particular, follow the applicable coding standards.

Instructions

This assignment is to be done individually.

Implement a C++ class template that manages and allows access to a fixed-size array. The type of item in the array and the number of items in the array should be specified by the client. Be sure to follow the coding standards. Standard 3A ("Requirements on template parameter types must be documented.") now applies. *You do not need to follow standards 3B, 3C, or 3D.*

- Name your class template "`KSArray`", so that, for example, a `KSArray` whose items have type `double` would be declared as `KSArray<double>`.
- Implement your class template in the file `ksarray.h`. Since everything in this file will be template, there should be no associated source file.
- Your class template should act as if it maintains a single array with a given size and value type.
- Include the following **functions**, and no others, in the public interface of your package:
    - Default ctor: This should create a `KSArray` of size 10. All items in the array should be default-constructed.
    - Dctor, copy ctor, move ctor, copy assignment, move assignment:
        - Be sure that the dctor frees any dynamically allocated memory.
        - The copy operations should create an entirely new copy of the data, so that modifying the copy does not change the original.
        - Write these functions as described in the slides (*Writing the Big Five*).
    - 1-parameter ctor: parameter is a non-negative integer giving the number of items in the array. All items in the array should be default-constructed.
    - Bracket operator: Given an integer subscript from 0 to $size - 1$ (where $size$ is the number of items in the array), return a reference to the proper item.
    - Member function `size`: no parameters. Returns the number of items in the array.
    - Member function `begin`: no parameters. Returns the address of item 0 in the array (think "iterator").
    - Member function `end`: no parameters. Returns the address of the item one-past the end of the array (think "iterator").
    - Operators "==" and "`!=`". Two `KSArray` objects with the same value type are equal if they have the same size and their corresponding items are all equal. Two `KSArray` objects with different value types cannot be compared.
    - Operators "<", "<=", ">", and ">=". As above, two `KSArray` objects with different

value types cannot be compared. For two `KSArray` objects with the same value type, this should be *lexicographic order*, which works like alphabetical order. So, when comparing `KSArrays` `a` and `b`, if `a[0] < b[0]`, then `a < b`. If `a[0] > b[0]`, then `a > b`. Otherwise, we proceed to `a[1]` and `b[1]`, and compare them similarly. If each item in `a` is equivalent to the corresponding item in `b`, and the size of `a` is less than that of `b`, then we should have `a < b`. Lastly, all of these operators should be consistent, so that, for example, `a > b` precisely when `b < a`, and `a >= b` when `!(a < b)`.

- Include the following **member types**, and no others, in the public interface of your package:
  - `value_type`: the type of each item in the array.
  - `size_type`: the type of the size of an array and also of an index into an array. *Be sure to make a good choice for what this type is!*
- A `const KSArray<T>` should be one that does not allow modification of the items in its array. A non-const `KSArray` should allow such modification. *Hint: This has implications for how you implement functions* `begin` *and* `end`*, as well as the bracket operator.*
- You may *not* use any C++ Standard Library **classes** or **class templates** in your implementation. (You may use simple types, like `std::size_t`, and functions or function templates, like `std::swap`, `std::copy`, `std::equal`, etc.)
- You may not create any new implicit type conversions. *Hint: 1-parameter constructors, "*`explicit`*".*
- As in Assignment 1, some parameter values may be illegal for some functions. You must deal with this somehow. The way you do it is up to you, but you must **document** how you did it.
- Avoid unnecessary duplication of code.

Example Code

Here is some code using `KSArray`. Some of it will not compile, as noted.

```
KSArray<int> ia(10);          // Array of 10 ints
KSArray<int> iax;             // Another array of 10 ints
KSArray<double> da(40);       // Array of 40 doubles
KSArray x(10);                // WILL NOT COMPILE; no template parameter
```

```
// Set all items (counter loop)
for (int c = 0; c < ia.size(); ++c)
{
    ia[c] = c * c;
}

// Print all items (iterator loop)
const int * iter;
for (iter = ia.begin(); iter != ia.end(); ++iter)
{
    cout << "Item :" << *iter << endl;
}

const KSArray<int> ia2(ia);   // Copy constructor
if (ia2 == ia)                // Condition should be true
    cout << "Equal!" << endl;

KSArray<double> da2;
da2 = da;                     // Copy assignment
da2 = ia;                     // WILL NOT COMPILE; different types

if (da == ia)                 // WILL NOT COMPILE; different types
    cout << "blah blah" << endl;
```

Test Program

A test program is available. This is a single program comprising two source files: `ksarray_test_main.cpp` and `ksarray_test_suites.cpp` If you compile and run the test program (unmodified!) with your code, then it will test whether your code works properly.

The test program requires `catch.hpp`, the single-header version of the “Catch” unit-testing framework.

Do not turn in the test program or the Catch framework.

Note: As before, the test program does not check *all* of the requirements of the assignment. Some of these cannot be checked by a test program.

Thoughts

- Once again, this is to be **high-quality** code.