CISS245: Advanced Programming
Assignment 13

Objectives:
1. Write a class template with member and non-member functions.

As always read the whole document carefully before diving into coding.

Q1.

The goal is to convert a (regular) class to a template class. The class chosen is our `vec2d` class.

We already have a `vec2d` class. Objects of this class models 2-dimensional vectors with `double` coordinates. Frequently in games (or scientific simulation), `float`s are sufficient. Note that `double`s can represent more real numbers than `float`s. Furthermore in many "simple" games such as 2-d games from the 80s, integer coordinates are enough and integer operation are a lot faster than `double` or `float` operations.

The goal of this question is to build a template 2-dimensional vector class, `vec2`. You should use the `vec2d` class from our previous assignment. With this class we can create 2-dimensional vectors of different numeric types (`int`, `float`, `double`) like this:

```
vec2< int > u(1, 2);         // u is a 2-d vector with integer coordinates
vec2< float > v(1.2f, 3.4f); // v is a 2-d vector with float coordinates
vec2< double > w(1.2, 3.4);  // w is a 2-d vector with double coordinates
```

In your `vec2` header file you should include three typedefs:

- `vec2i` which is an alias for `vec2< int >`
- `vec2f` which is an alias for `vec2< float >`
- `vec2d` which is an alias for `vec2< double >`

Of course this depends on `vec2`, so these typedefs should be after the `vec2` class.

With these typedefs the above examples become

```
vec2i u(1, 2); // u is a 2-d vector with integer coordinates
vec2f v(1.2f, 3.4f); // v is a 2-d vector with float coordinates
vec2d w(1.2, 3.4); // w is a 2-d vector with double coordinates
```

Note that the length function, `len()`, must return a `double` regardless of the template type parameter.

Also, note that the argument for `operator[]` is either 0 or 1. If a value other than 0 or 1 is given, then you must throw a `ValueError` exception. This class should be included at the the top of your `vec2.h` file, i.e.,

```
#ifndef VEC2_H
#define VEC2_H

class ValueError
{};

...
#endif
```

and the following will catch a `ValueError` object:

```
vec2f v(1, 2);
try
{
    std::cout << v[42] << '\n';
}
catch (ValueError & e)
{
    std::cout << "caught ValueError object\n";
}
```

Test your code thoroughly. The test file must be named `testvec2.cpp`. (The test cases you should include should be very similar to the test code for `vec2d`.)

WARNING: I've already mentioned this. You must keep all the code for templates in header files.

This illustrates a very common approach to developing a class template. You usually first pick a special case of the template and develop it as a concrete class before you generalize it to a class template. In many cases once the specific concrete case works correctly, the generic template case works with very little extra work. On the other hand, if you begin with the class template directly, you usually end up getting lots of convoluted error messages. In general error message from template classes requires a lot more carefully reading. Hence it's more time consuming to work directly on a class template.

NOTE: You can assume the user will *not* perform mixed type computations such as `vec2< int >(1, 2) + vec2< double >(1.1, 2.2)` which involves `int` and `double` type values. The normalization of a vector with `double`s is a vector of `double`s and the normalization of a vector with `float`s is a vector of `float`s.