5 Templates

5.1 Preamble

The exercises below are intended to provide a means to test your understanding of the programming-related material covered in the course. It is highly recommended that you work through these exercises as you cover the corresponding topics in the video lectures. By doing this, you will greatly strengthen your understanding of the material in these video lectures, which will greatly reduce the amount of pain and suffering required to complete the programming assignments in the course. In exercises that require the building (i.e., compiling and linking) of code, the CMake tool should be used for this purpose. For additional information about these exercises, refer to Section 1 of this document.

5.2 Topics Covered

These exercises cover templates, including such things as: function templates, class templates, and qualified dependent names.

5.3 Exercises

1. **[function template]**
   Write a template function called `max3` that meets the following specifications:
   - The function takes three parameters `a`, `b`, and `c`, all having the type `T` (where `T` is arbitrary).
   - The return value of the function has type `T` and should correspond to the maximum of `a`, `b`, and `c`.

   You may safely assume that the type `T` provides the basic relational operators (e.g., less than, greater than, etc.). Test your code using the function `myTest1` given below. (To save typing, this function can be downloaded from the course web site in the file `myTest1.cpp`.)

```
void myTest1()
{
    {
        std::cout << "Enter three integers: ";
        int a;
        int b;
        int c;
        std::cin >> a;
        std::cin >> b;
        std::cin >> c;
        if (std::cin) {
            std::cout << "The greatest of these three integers is: "
              << max3(a, b, c) << "\n";
        }
    }

    {
        std::cout << "Enter three words: ";
        std::string a;
        std::string b;
        std::string c;
        std::cin >> a;
        std::cin >> b;
        std::cin >> c;
        if (std::cin) {
            std::cout
              << "The lexicographically greatest of these three words is: "
              << max3(a, b, c) << "\n";
        }
```

```
    }
}
```

2. **[class template]**
   (a) Develop a class template, called `Array`, that represents a one-dimensional fixed-size array of objects of some fixed (but arbitrary) type. The class should meet the following specifications:
   - The class template takes the following two parameters (in order):
     i. `T`, the type of each element in the array; and
     ii. `size`, an **`int`** specifying the number of elements in the array.
   - The class provides a single private data member called `data_`, which is an array containing `size` elements of type `T`.
   - The class should overload **`operator`**`[]` using public member functions so that array indexing operations work in the manner that one would normally expect. For example, if `array` is an object of type `Array` and `i` is of type **`int`**, then `array[i]` would be a reference to the `i`th element in the array.

   In addition, **`operator`**`<<` should be overloaded in order to allow objects of type `Array` to be written to an output stream (i.e., `std::ostream`). That is, a stream inserter should be provided for the `Array` template class.

   (b) Write a program to test your `Array` template class. Your test code should consider at least two different choices for the template parameters. In addition, your code should also use the `myTest2` function given below. (To save typing, this function can be downloaded from the course web site in the file `myTest2.cpp`.)

```
void myTest2()
{
    Array<int, 10> array1;
    std::cout << array1 << "\n";
    for (int i = 0; i < 10; ++i) {
        array1[i] = 2 * i;
    }
    std::cout << array1 << "\n";
    const Array<int, 10>& r1 = array1;
    std::cout << r1 << "\n";

    Array<std::string, 3> array2;
    std::cout << array2 << "\n";
    array2[0] = "hello";
    array2[1] = "world";
    array2[2] = "goodbye";
    std::cout << array2 << "\n";
    const Array<std::string, 3>& r2 = array2;
    std::cout << r2 << "\n";
}
```

   (c) Modify the `Array` class template so that the template class parameters `T` and `size` default to **`int`** and 2, respectively. Test your modified code by using the `Array` class template without explicitly specifying any parameters to the template.

3. **[qualified dependent names]**
   Consider the program given in the listing below. (To save typing, the source code for this program can be downloaded from the course web site in the file `qdn.cpp`.) Confirm that the program does not compile. Find the error in the source code and correct it.

```
// A class used to represent a rational number
// (i.e., a number that is the ratio of two integers).
class RationalNumber
{
public:
    // The integer type used to represent each of the numerator and
    // denominator of the rational number.
    typedef long long NumberType;
```

```
private:
    NumberType numerator_;   // The numerator value.
    NumberType denominator_; // The denominator value.

    // Much more irrelevant code deleted...
};

// A template class that uses the RationalNumber class.
// The class template takes a single parameter that is a type to be used
// for representing rational numbers.
template <class RationalNumberType>
class SomeClass
{
public:
    // Make an abbreviated name for RationalNumberType::NumberType
    typedef RationalNumberType::NumberType NumType;

private:
    // A data member used to store the numerator or denominator of
    // some rational number.
    NumType someValue_;

    // Much more irrelevant code deleted...
};

int main(int argc, char** argv)
{
    SomeClass<RationalNumber> x;
    return 0;
}
```