

689: Generic Programming — Assignment 4

November 16, 2004

IMPORTANT! This is an individual assignment, not to be done in your project groups.

1. Consider the metafunction (from slides set 6) that maps binary numbers to base 10 integers. The definition is less than ideal, since the metafunction accepts input that contains digits other than 0 or 1, and still produces results for such invocations. Add error checking to the metafunction, so that invocations with such input is rejected.
2. Let A be the following set of some of the C++ arithmetic types: { **char**, **int**, **long**, **float**, **double**, `complex<float>`, `complex<double>` }.

Consider the partial order on A :

- **char** < **int** < **long** < **float**
- **float** < **double**
- **float** < `complex<float>`
- **double** < `complex<double>`
- `complex<float>` < `complex<double>`

Write a metafunction `promote` that takes a `RANDOM ACCESS SEQUENCE` whose elements are types from A , and returns a new sequence of the same length, where each element has been promoted to the lowest upper bound of the types of the input sequence. For example, the following should be true:

```
mpl::equal<
  promote< mpl::vector<char, double, int> >::type,
  mpl::vector<double, double, double>
>::type::value
```

3. Define a generic `add` function (or a set of generic functions) that takes three `std::vectors` as its parameters; first two are used as input, the third as the output. The function should perform an elementwise addition of the two input vectors, and write the result into the output vector.

The element type of each vector can be different, as long as the element types of the output vector are greater than the element types of the input vectors; here 'greater' refers to the partial order defined in exercise 2. The `add` function should work for *exactly* those calls where the parameter types satisfy this condition, and not be defined for any other types. For example:

```
std::vector<char> x;
std::vector<long> y;
std::vector<long> z;
// some code that fills vectors
add(x, y, z); //ok
add(x, x, y); //ok
add(y, z, x); //add not defined
```

The `enable_if` template may prove useful, as well as your work with problem 2.

4. Consider the following function:

```
template <class IterA, class IterB, class C>
C dot_product(IterA a, IterA aend,
              IterB b, C init) {
    while (a != aend) {
        init += *a++ * *b++;
    }
    return init;
}
```

Write a concept description for the *single* concept `DOTPRODUCTABLE` that expresses the requirements of `dot_product` for all of its template parameters. Reuse standard library concepts if possible. Is your concept description minimal? (Are there any types that do not model the concept, but would still work correctly with `dot_product`?)

What to turn in

Follow the directions from assignment 1, except now name your tar-ball as `<your_name>.tar.gz`.