C++ Intermediate Exercises

Question 1
STL - Filters.cpp

Write a function that generates a range of integers 1 .. 50 and stores them in a vector and then filters out all the odd numbers from the vector. Print out the resulting range of even numbers.
Use one of the STL algorithms to perform the filtering.

Question 2
STL - Comprehensions.cpp

Write a template that takes a vector<T> and applies a function to each element in the collection and storing the results in a new vector<T>.
Hint: use the transform algorithm.

For example,

```
int main()
{
	vector<int> v = { 2, 4, 6, 8, 10, 12 };
	// apply cube() to each element and store in v2
	copy(v2.begin(), v2.end(), ostream_iterator<int>(cout, ","));
	cout << endl;
}
```

would produce output:

```
8,64,216,512,1000,1728,
```

and

```
int main()
{
	vector<double> v = { 2.5, 3.5, 4.5, 5.5, 6.5 };
	// apply square() to each element and store in v2
	copy(v2.begin(), v2.end(), ostream_iterator<int>(cout, ","));
	cout << endl;
}
```

produces:

```
6.25,12.25,20.25,30.25,42.25,
```

Question 3
chrono - Calendar.cpp

Use the ***chrono*** Calendar library to print a calendar for a given year (e.g 2023). Your output should look like:

```
CALENDAR for 2023


January
=======
 Mon Tue Wed Thu Fri Sat Sun
                           1
   2   3   4   5   6   7   8
   9  10  11  12  13  14  15
  16  17  18  19  20  21  22
  23  24  25  26  27  28  29
  30  31

February
========
 Mon Tue Wed Thu Fri Sat Sun
           1   2   3   4   5
   6   7   8   9  10  11  12
  13  14  15  16  17  18  19
  20  21  22  23  24  25  26
  27  28

March
=====
 Mon Tue Wed Thu Fri Sat Sun
           1   2   3   4   5
   6   7   8   9  10  11  12
  13  14  15  16  17  18  19
  20  21  22  23  24  25  26
  27  28  29  30  31

etc
```

Question 4
Filesystem - SearchForStringInFiles.cpp

Use the Filesystem library to implement a function that recursively searches a set of files from a given root direct for files of a set extension (e.g. ".txt"). For each file found then check if the file contains a given string (e.g. "Magic").

Use the following main program to test your work:

```
int main()
{
   string textToFind = "Magic";
   string extension = ".txt";
   fs::path root_path("resources");
   vector<fs::path> result = findFiles(root_path, extension);
```

```
   for(unsigned i = 0; i < result.size(); i++)
   {
      bool found = search_for_string(result[i], textToFind);
      if(found)
      {
          cout << "Found: " << textToFind << " in file: "
              << filesystem::canonical(result[i]).string() << endl;
      }
   }
}
```

Question 5
Templates - CustomComparators.cpp

Consider the class below:

```
class Person
{
private:
   string name;
   string address;
   int age;
public:
   Person(string name, string address, int age)
      :name(name), address(address), age(age) {}

   string getName()    const  { return name; }
   string getAddress() const  { return address; }
   int    getAge()     const  { return age; }
   void print() const
   {
      cout << "[" << name << ","
                  << address << ","
                  << age << "]" << endl;
   }
};
```

Each person object has three fields. Create a collection of objects using the code below and **use the sort algorithm to sort the collection three times, once for each field**:

```
int main()
{
   vector<Person> theList;

    theList.push_back(Person("Smith", "London", 43));
    theList.push_back(Person("Jones", "Cardiff", 51));
    theList.push_back(Person("Lee",   "New York", 24));

   // sorting goes here
}
```

The sort algorithm takes 3 parameters:

```
void sort(RandomAccessIterator first,
          RandomAccessIterator last,
          Compare functor);
```

where the third parameter is defined as a functor class. Make the functor into a template with the signature:

```
template <typename T, int N>
class Compare
{
   // ...
}
```

where N defines the field on which to sort (N = 1, 2 or 3).

Question 6
C++11 - Lambdas.cpp

Consider the code below:

```
int main()
{
   cout << do_operation1("+", 5, 7) << endl;
   cout << do_operation1("-", 15, 3) << endl;
   cout << do_operation1("*", 2, 6) << endl;
   cout << do_operation1("/", 24, 2) << endl;

   cout << do_operation2("+", 5, 7) << endl;
   cout << do_operation2("-", 15, 3) << endl;
   cout << do_operation2("*", 2, 6) << endl;
   cout << do_operation2("/", 24, 2) << endl;
}
```

Your job is to implement the two do_operation functions subject to the following restrictions:

do_operation1():

a) use an *unordered_map* in the function to store 4 lambdas.
b) the 4 lambda functions should implement functions to add, subtract, multiply and divide for two input parameters. e.g.

```
[ ](double a, double b) { return a + b; };
```

c) use the first parameter("+", "-" etc) to invoke the appropriate lambda and return the result of the calculation.

do_operation2():

a) reimplement do_operation1() to use closures such that the lambdas look like:

```
[&a, &b]() { return a + b; };
```

Note that you will need to use the <functional> header to provide bindings that work with closures.

If it helps, here is the Python solution:

```
def do_operation(op, a, b):
    return {    '+':    lambda: a + b,
                '-':    lambda: a - b,
                 '*':   lambda: a * b,
                '/':    lambda: a / b
            }[op]()
```