Santa Clara University
School of Engineering

# CSEN 79L: STL Vector and Algorithms

In this lab, we practice using STL vector and the sorting/searching algorithms.

## Background

C++ Standard Template Library (STL) has become an integral part of the language, like the "C Library" (cstdlib) for "C". It pays to be familiar with its features, particularly the rich set of containers and a rich set of library functions.

The vector container, based on template, accepts one parameter of the element type. Like almost all other STL containers, it supports iterator for ease of traversal. Further, it is well integrated with the searching and sorting facilities.

There were several searching and sorting facilities, the popular ones are `find`, `lower_bound`, and `upper_bound`. They all have several variations in their interfaces. The most interesting one being the acceptance of a "less-than" comparator for custom searching and sorting.

If not provided, these functions use the standard "<" operator to compare elements within the vector. These STL search/sort functions are similar in the way they do not rely completely on the standard operators. They also accept a customer "comparator" that defines "less than" or "equal to" as the programmer sees fit. The comparator can be a function or a ***functor*** as the comparator. If a function is provided, vector calls the function with 2 elements within the vector and expect either true or false as the return value. Note that a function can be named as a normally defined function. It can also be unnamed as a lambda, but that's beyond the scope of this exercise.

A functor, formerly a function object, is an object (which is an instantiation of a class) that behaves like a function. How? By providing the "function call operator" as a public member within the class. Put different, if a class has an "function call operator" then its objects can behave like a normally defined function.

For these STL algorithm functions, the specific function call operator must accept two vector elements and return true/false for the comparison logic.

## Examples

We have an sample code to illustrate how STL searchy/sort functions work in `stldemo.cxx`.

If we have a vector of integer "`vlist`", we can try to search for a value (`target`) within it by calling find.

```
vector<int> vlist ={…};
auto it = find(vlist.begin(), vlist.end(), target);
```

Vector uses the equal-to operator (==) to compare within the vector and return the first element, as an iterator, to the caller. If the returned iterator is the same as "`vlist.end()`", the element was not found. It's common to reuse the iterator to find more occurences.

In the example file stldemo.cxx, we wrote two trivial functions that are the "less than" and "equal to" comparitors.

```
bool lessComp(const int a, const int b) { return a < b; }
```

```
bool eqComp(const int a, const int b) { return a == b; }
```

It also provided a trivial class to generate a functor.

```
struct CmpLessThan {
     const bool operator()(int x, int y) {return x < y; }
};
```

And you can see how they are used in the example file.

Note that "`lower_bound`" returns the first element that is greater than or equal to the compared value. The "`upper_bound`" returns the first one that is "greater than" the value.

## Assignment

The file `data.txt` lists name, birth year/month/day, the height in inches and weight in pounds. The code in `record.cxx` and `record.h` provided the input/output interfaces to compatible to the file format. The `main.cxx` has the code fragment to read the file into a `std::vector` of Record elements.

Implement a flavor of `operator==` and use `std::find` to list all those who were born in March.

Implement another one and use `std::find` to see if "Lily Liu" exists on the list.

Implement a flavor of `std::operator<` and use `std::sort` to arrange the list in the alphabetically by last name.

Lastly, implement a functor and also `std::sort` to arrange the list in ascending order of their BMI (Body Mass Index) value. BMI is defined by the weight divided by the square of the height in metric. For the US units, use pound and inches and multiply the result by 703.