C++ Notes: Algorithms: STL sort for arrays

Never write your own sort! Use the the sort in the Standard Template Library (STL). The STL has sorts that are efficient and well tested.

Basic syntax for calling `sort`

When calling the STL sort, you need to pass two parameters: the address of the first element to sort, and the address of *one past* the last element to sort. The address is used for iterating across array elements. For other data structures (eg, a `vector`) you will have to do something a little different, but for arrays we can simply express the beginning and ending points with the array name and the addition of an integer. For example,

```
1  // sorting/stl-sort-array.cpp - Demo STL sort of array.
2  // Fred Swartz - 2003-09-26
3
4  #include <iostream>
5  #include <algorithm>
6  using namespace std;
7
8  int main() {
9      int a[7] = {23, 1, 33, -20, 6, 6, 9};
10
11     sort(a, a+7);
12
13     for (int i=0; i<7; i++) {
14         cout << a[i] << " ";
15     }
16
17     return 0;
18 }
```

This prints the sorted values.

```
-20 1 6 6 9 23 33
```

Include header

The `<algorithm>` header must be included.

Using plus to compute the address of an array element

An array variable is the address of the first element (eg, `a` is the address of `a[0]`), and the address of any element may be computed by adding an integer to the address of the first element. In this example, `a` is the address of the first element, and `a+7` is the address of the eighth element (ie, the address of `a[7]`). How can we use `a+7` if last element of the array is `a[6]`? See below.

The element past the end

The `sort()` function requires the end to be indicated with the address of the element `beyond` the last element that is to be sorted. Even if there is no element in the array, the address of this hypothetical element can be computed. Don't worry, `sort()` never tries to reference data at that position, it just uses that address as a upper limit.

Sorting predefined types

There is no problem sorting any of the predefined types (eg, `int`, `float`, `char`, ...).

Sorting class (struct) types

If you define a new type using `struct` or `class`, `sort()` has no idea how to compare two values. For example, if a new `Student` class is defined, what would it mean to compare two elements?

Defined as class	Equivalent struct declaration

`class Student {`
`  public:`
`  int id;`
`  string first_name;`
`  string last_name;`
`  float gpa;`
`};`
	`struct Student {`
	`  int id;`
	`  string first_name;`
	`  string last_name;`
	`  float gpa;`
	`};`

For the following discussion, `class` will be used instead of `struct`, but they are completely equivalent except that all members of a `struct` default to `public`.

Comparison is not defined by default for class objects

There are very few operators which are defined by default for user-defined classes. Assignment is defined, but none of the comparison operaters are defined.

For example, if we defined two students, what would it mean to compare them?

```
Student betty;
Student bob;
. . .   // assign some values.
if (betty > bob) {  // ILLEGAL, ILLEGAL, ILLEGAL
```

`sort()` needs comparison and assignment

The STL `sort()` method needs to compare elements and assign them. It uses the less-than (<) operator to

compare, but less-than isn't defined for user types.

C++ will perform assignment between classes/structs, so for simple structs that don't do dynamic allocation that generally isn't a problem.

You must define less-than for `sort()`

Overloading less-than is fairly simple. For the `Student` class let's define the comparison to be for the `gpa` field. We could also define it to be for the `id` or the `name` or whatever just as easily.

```
bool operator<(const Student& a, const Student& b) {
    return a.score < b.score;
}
```

The keyword `operator` is prefixed to the operator and the two parameters are passed as `const` (they won't be changed) reference parameters. A `bool` value is returned. After this function is defined, the STL `sort()` function may be used.

This is a very brief summary of how to define operators. See Operator Overloading